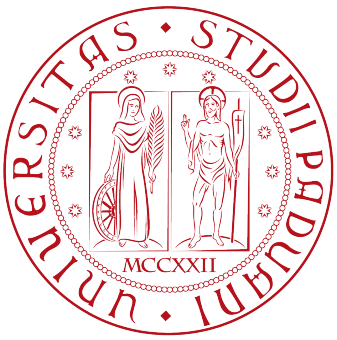
Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in:
ICT for Internet and Multimedia

Automatic Labelling of 3D Motion Capture Markers using Neural Networks



Relatore:
Prof. Michele Rossi
Correlatori:
Prof. Zimi Sawacha
Ing. Gianmarco Bortolami
Dott. Diego Crovato

Candidato:
Edoardo Monaco

Anno Accademico 2021/2022
Data di Laurea 11 Aprile 2022

Abstract

The work focuses on the development of a machine learning framework based on neural networks, which is able to predict the labels of 3D markers from a motion capture system. This should be implemented in an existing application devoted to the biomechanical analysis in the medical and sports field. Starting from a dataset of 3D files acquired in field and in laboratory from an heterogeneity of patients and athletes, containing specific movements, the purpose is to speed up the process of analysis during which somebody until now proceeds with a manual labeling of the 3D cloud of points, before the data can be analyzed. The neural networks framework implemented is based on a strict pre-processing and post-processing voted to clean the data in order to get a better result and to handle the presence of missing and extraneous markers, while the core of the algorithm is a LSTM Neural Network. The training performed on the network on a first computable sub-set has been a success, revealing a test accuracy of 95%.

Il lavoro si concentra sullo sviluppo di un framework di machine learning basato su reti neurali, in grado di predire le assegnazioni di marcatori 3D da un sistema di motion capture, per un'applicazione esistente dedicata all'analisi biomeccanica in campo medico e sportivo. Partendo da un dataset di file 3D acquisiti in campo e in laboratorio da una eterogeneità di pazienti e atleti, contenenti movimenti specifici, lo scopo è quello di accelerare il processo di analisi attualmente svolto manualmente che prevede un etichettatura manuale dei dati tridimensionali, prima che i dati possano essere analizzati. Il framework di reti neurali implementato si basa su una rigorosa pre-elaborazione e post-elaborazione votata alla pulizia dei dati per ottenere un risultato migliore e per gestire la presenza di marcatori mancanti ed estranei, mentre il cuore dell'algoritmo è una serie di reti neurali LSTM. Il training della rete performato su di un primo sub-set è stato un successo, rilevando un'accuratezza in fase di test del 95%.

Index

1 Introduction

1.1 Motion Capture

Motion Capture is defined as a technology for digitally recording specific movements of a person or of an object and translating them into computer-animated images. The first idea behind it, has been designed during the 19^{th} century, to examine how animals and humans move. Before its introduction in the world of cinema (late 20^{th} century), Motion Capture has been used as an analysis tool in biomechanics, sports and education [1].

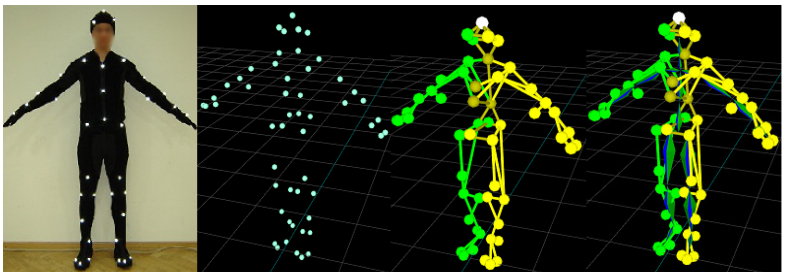


Figure 1.1: Motion Capture example.

Since the 20^{th} century, the performer has been required to wear markers near each joint, as shown in the picture above, to identify movement based on the positions or angles between the markers. Those markers can be of different types, like inertial, LED, magnetic, reflective, acoustic or a combination of these.
Since the beginning of the 21st century, and due to the rapid growth of technology, new methods have been developed. Most modern systems can extract the performer's silhouette from the background. Then, all joint angles are calculated by inserting a mathematical model into the silhouette. For movements where it is not possible to see a change in the silhouette, there are hybrid systems available that can do both (marker and silhouette), but with fewer markers [2].
The markers are critical for most motion capture acquisitions, and one of the main features associated with them is the labeling, which determines which marker is associated with which part or joint of the body.
In simple acquisitions using adhesive tape or fabric markers, in which the data is collected using normal cameras, as in other cases, the labeling of the markers is often performed manually. This process is time consuming and it is subject to human error.

1.2 Motion Capture for Biomechanical Analysis

Motion capture technology has been introduced in the life sciences during the early 1970s for gait analysis, and today biomechanical research still remains the most common application of the technology.
Today, the technology is at the forefront of clinical motion research, including helping to rehabilitate injured military personnel and improving the performance of world-class athletes. Motion capture is highly relevant in universities, research centers and hospitals [3].

Figure 1.2: Motion Capture in Sport Field.

Optical motion capture systems are used in a wide range of applications, for example, gait analysis provides clinicians with a greater understanding of lower limb movement. It also offers significant advantages in motor control and neuroscience, enabling important advances in the treatment of patients with complex neuro-musculoskeletal injuries, including cerebral palsy and myelomeningocele.
The tools involved in the motion capture for Bio-Analysis are the following:

- Marker-based optical trackers.
- Inertial sensor-based systems.
- Markerless systems.
- Sophisticated human body models.

- Computer vision.

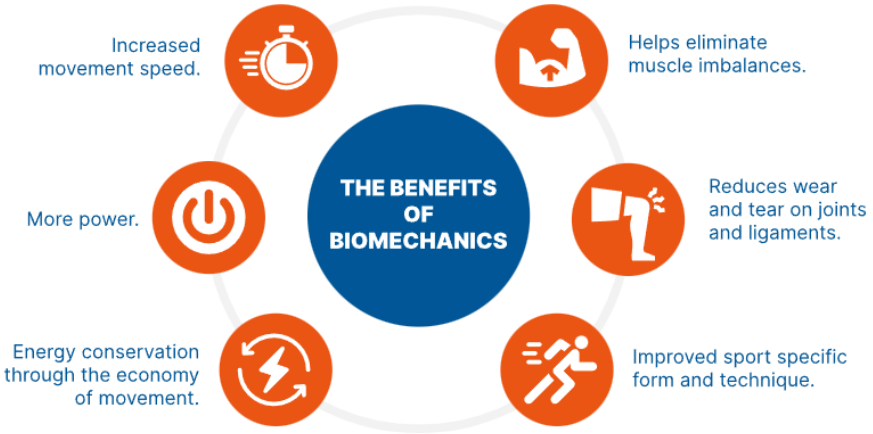


Figure 1.3: Biomechanical Motion Capture fields of interest [4].

Over the last few years, inertial motion capture technology, which allows subjects' movements to be tracked and recorded outside on the field rather than in the lab, is also actively used by coaches and athletes in a wide range of sports to improve performance. Being able to measure and evaluate players and athletes allows coaches to not only improve technique, but also correct actions to prevent potential injury. It is also key to accelerating athletes' return to play after injuries, ensuring that recovery occurs safely and minimizing the possibility of re-injury.

Thanks to technological development, motion capture systems became more accessible, in particular if considered the availability of affordable inertial devices and apps that allow you to display data in a more digestible way. For those working in the broad field of biomechanics, motion capture allows users to have more data-driven methods, to understand in a more efficient way how their subjects move.

Growing advances in technology are providing decision makers within the medical space with access to more data than ever before, and in turn are helping to improve the quality of not only patient care, but the quality of life for patients and athletes around the world [3].

1.3 The Goal

The project takes place in the context of an internship at B.B.S.o.F., a spin-off of Padova University, starting from one of their applications "Track-on-Filed", that

works on dynamic sport-specific testing with indoor and outdoor technical apparel, at the athletes' filed.
The goal is to introduce inside the application an Artificial Intelligence approach able to label automatically the markers of the motion capture, avoiding a long and time consuming work from the lab staff.
This has been possible thanks to an advanced and public state of art, from which it is possible to derive the best techniques for approaching the problem, described in section 2.1.

1.4 Structure of this Thesis

In the next chapters will be introduced all the tools necessary to better understand the algorithm implemented.
The chapter 2 presents the state of the art for both the auto-labeling for the motion capture, and for the artificial intelligence tools used, the LSTM neural network.
The chapter 3 is dedicated to those biomechanical tools that are necessary to comprehend the structure of the data and the file format that are used in biomechanics, and other features related the motion capture world associated to the Digital Health.
Chapter 4 presents the neural networks tools, from the vanilla recurrent neural network, to the advanced long short memory term network and its architecture.
Furthermore, in chapter 5 is explained the auto-labeling algorithm implemented, from the analysis of the dataset to the pipeline of the whole algorithm itself.
Finally, chapter 6 shows the results and the difficulties found in the tuning and training of the algorithm.

2 State of the Art

This chapter presents the state of the art connected to the main features covered in this work, the auto-labeling for the motion capture analysis and the history, fairly recent, of the artificial intelligence tools used in the algorithm, the LSTM Neural Network.
Concerning the LSTM part, only the details about the development, the features introduction and the most famous use of them during the years are presented in this section, while the technical features are explained in section 4.2.1.

It is interesting to see how the problem of labeling is a common point in the world of biomechanics, and how during the years, the work has been taken into consideration and improved from the previous ones, without a substantial overhaul.

2.1 State of Art of Automatic Labeling for Motion Capture

The typical pipeline for a Motion Capture analysis starts from the recording phase, in which the movement of the passive reflective markers, attached to the body according to a predefined arrangement of markers, is followed by multiple high-resolution infrared cameras able to collect both spatial and temporal features.
Then, the 3D positions of the markers are computed from the 2D data recorded by each of the cameras calibrated by triangulation. The result of this process is a set of 3D trajectories listed in random order. Each trajectory represents the movement of a single marker in terms of its 3D position over time.
The first step after the registration phase is to label each trajectory, thus assigning it to a specific body position. This process can be very time-consuming and it is a critical point for many motion capture systems. In addition, this manual process is susceptible to user error.
Labeling is more challenging when one or more markers are occluded over time, dividing in this way the movement of these markers into multiple trajectories. Commercial motion capture software has designed facilities to reduce the amount of manual work at this stage. In these facilities, a model can learn the geometry of that participant's body and apply it to label the subsequent measurements. However, a manual initialization is typically required and during this procedure one or more motion capture sequences must be manually labeled for each participant [5].
Motion capture labeling usually includes two main phases: the initialization phase where initial matches are established for the first frame, and the tracking phase

which can be defined as the maintenance of the trace of labeled markers in the presence of occlusion, ghost markers, and noise.

Multiple approaches have have been introduced during the years to facilitate the tracking of manually initialized motion capture data: Holden [6] proposed a deep feed-forward denoising network that produces the joint positions directly from the corrupted markers.
Herda et al. [7] proposed an approach aimed to increase the robustness of optical markers during visibility constraints and occlusions. To do this they used kinematic information provided by a generic human skeletal model.

Yu et al. [8] introduced an online motion capture approach suitable for multiple subjects able to retrieve missing markers. In particular they used the standard deviation of the distance between each pair of markers to group the markers into a number of rigid bodies. After assembling the rigid bodies, they labeled the markers using a motion model for each marker and a structural model for each rigid body.

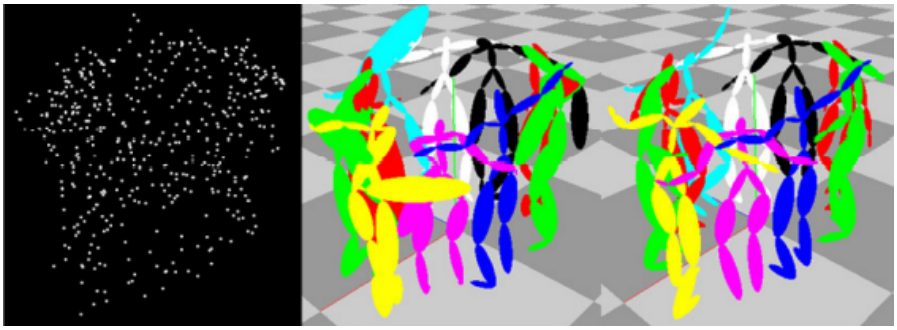


Figure 2.1: Snapshot of an input frame - ten persons are hugging [8].

Loper [9] proposed a refinement of marker placement by optimizing the parameters of a statistical model of the body thanks to a generative inference process. Another group of approaches seeks to minimize user intervention by also automating the initialization phase.
Holzreiter [10] introduced neural networks to estimate the positions of ordered markers from a shuffled set. The labeling of the markers was performed by pairing the estimated positions of the markers with the shuffled set searching for the closest neighbor marker.

Meyer et al [11] estimated the skeletal configuration using least-squares optimization and the skeletal model to automatically label markers. They applied the Hungarian method for optimal observation assignment to markers, requiring each subject to assume a T-pose to initialize the skeletal tracker.

Schubert et al. [12] improved upon their approach by designing a pose-free initialization phase, searching a large database of poses.
Han et al. [13] and Maycock et al. [14] proposed auto-labeling approaches designed specifically for hands. Han et al. [13] proposed a self-labeling of markers in optical MoCap by learning by permutation to label hand markers by formulating the task as a keypoint regression problem. They rendered markers locations as a depth image and using a convolutional neural network that produces the estimated 3D labeled positions of the markers, and they used a bipartite matching method for the labeling.

Maycock et al [14], in order to be able to filter out unrealistic postures, used inverse kinematics, and then computed the assignment between nodes and 3D points using the Hungarian method in an adapted way. Although their approach was focused on hands, it can be applied to human body motion.

Ghorbani et al. [5] have considered that problem as a classification process in which shuffled markers from a permutation matrix are sorted to recover the correct order. They proposed a framework for automatic labeling of markers that first estimates a permutation matrix for each individual frame using a differentiable permutation learning model, and then uses temporal consistency to identify and correct the remaining labeling errors. The Hungarian Algorithm is used again, as in most of the algorithms proposed during the last years (image 2.2).

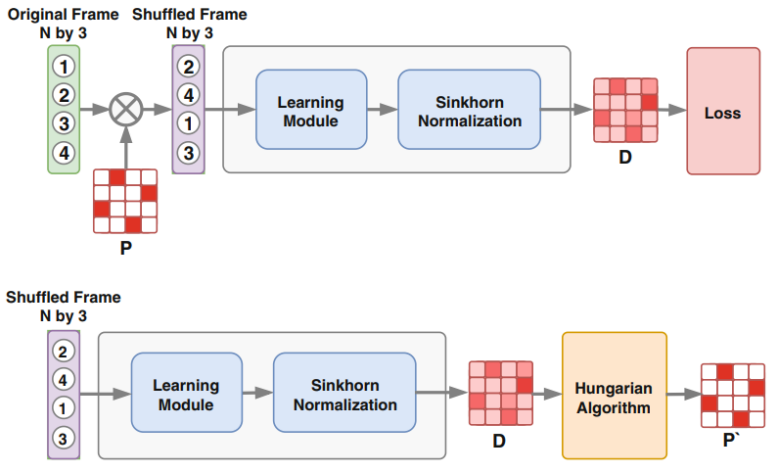


Figure 2.2: Learning and Running phases [5].

The last work considered is from Clouthier et al. [15] which proposed a learning-based algorithm for motion capture marker labeling that can be trained on the measured or simulated trajectories of the markers. The core of the algorithm is a deep neural network including recurrent layers trained on the measured or

simulated trajectories of the markers. Labels are assigned to the markers using the Hungarian algorithm as in some of the previous cases and a predefined generic set of markers is used to identify and correct mislabeled markers. The algorithm has been trained and tested on both measured motion capture and simulated data. A function of transfer learning has been introduced too, to update the neural network weights based on labeled motion capture data evaluated.

The state of art described has been taken into consideration during the work of this thesis, and it has been implemented with numerous changes inside the request of the company the work has been done for.

2.2 History and Development of LSTM Neural Networks

Long Short Term Memory Networks has been firstly proposed by Sepp Hochreiter [16] and Jürgen Schmidhuber [17] in the latest years 20th century, by introducing Constant Error Carousel (CEC) units. In particular LSTMs, compared with recurrent neural networks, were addressed to avoid the vanishing gradient problem. Few years later Felix Gers and his advisor Jürgen Schmidhuber and Fred Cummins introduce the forget gate (also called "keep gate") into the LSTM architecture, allowing the LSTM to reset its state [18].
Then Gers & Schmidhuber & Cummins added peephole connections (connections from the cell to the gates) in the architecture [19].
In 2013 LSTM networks has been an essential component of a network that achieved a record 17.7% phoneme error rate on the classic TIMIT natural speech dataset [20], and in 2014 Kyunghyun Cho et al. proposed a simplified variant called Gated recurrent unit (GRU) [21].
In 2015 Google started using an LSTM Network for speech recognition on Google Voice [22], and the next year it started using an LSTM to suggest messages inside the "Google Allo" conversation app[53]. In the same year, Google released a Neural Machine Translation system able to improve Google Translate using LSTM Neural Networks [23].
Apple announced at its Worldwide Developers Conference in 2016 that it would begin using LSTMs for rapid typing in the iPhone and for Siri [24].
Amazon released Polly, which generates the voices behind Alexa, using a two-way LSTM for text-to-speech technology.
Facebook performs about 4.5 billion machine translations every day using long short term memory networks [25].
From Michigan State University, IBM Research, and Cornell University, researchers

published a study for the Knowledge Discovery and Data Mining (KDD) conference. Their study describes a new neural network that performs better on some data sets than the widely used long short term memory neural network.
Microsoft reported achieving 94.9% recognition accuracy on the Switchboard corpus, which incorporates a vocabulary of 165,000 words. The approach used an LSTM based on the dialog session [26].
In 2019 Researchers at the University of Waterloo proposed a correlated RNN architecture representing continuous windows of time. It was derived using Legendre polynomials and outperforms LSTM on some memory-related benchmarks [27].

3 Biomechanical tools for 3D Motion Capture Analysis

In the first chapter the Motion Capture role in the biomechanical analysis has been briefly presented. In the following chapter will be shown the fundamental technologies, widely used during this work, for the analysis and collection of data. It will also be presented the software and the company this work has been done for.

3.1 Opensim

OpenSim is a powerful and freely available tool for motion modeling and simulation.

Figure 3.1: OpenSim Logo.

It is an extensible, easy-to-use software package built on decades of knowledge of computational modeling and simulation of biomechanical systems. The study of movement draws from and contributes to diverse fields, including biology, neuroscience, mechanics, and robotics and is based on the interaction of complex neural, muscular, and skeletal systems [28].
OpenSim combines methods from these fields to create fast and accurate motion simulations, enabling two key tasks. The software firstly is able to calculate variables that are difficult to measure in the experimental field, such as muscle-generated forces and tendon elongation and recoil during movement. Secondly, OpenSim is able to predict new movements from motor control models, such as kinematic adaptations of human gait during loaded or prone walking.
The changes in musculoskeletal dynamics can be generated by different factors, like surgery or the implementation of a human-device interaction. These simulations provided by the framework have a key role in several applications, including the design of implantable mechanical devices to improve human grasp in individuals

with paralysis.

OpenSim's design allows computational scientists to create new, state-of-the-

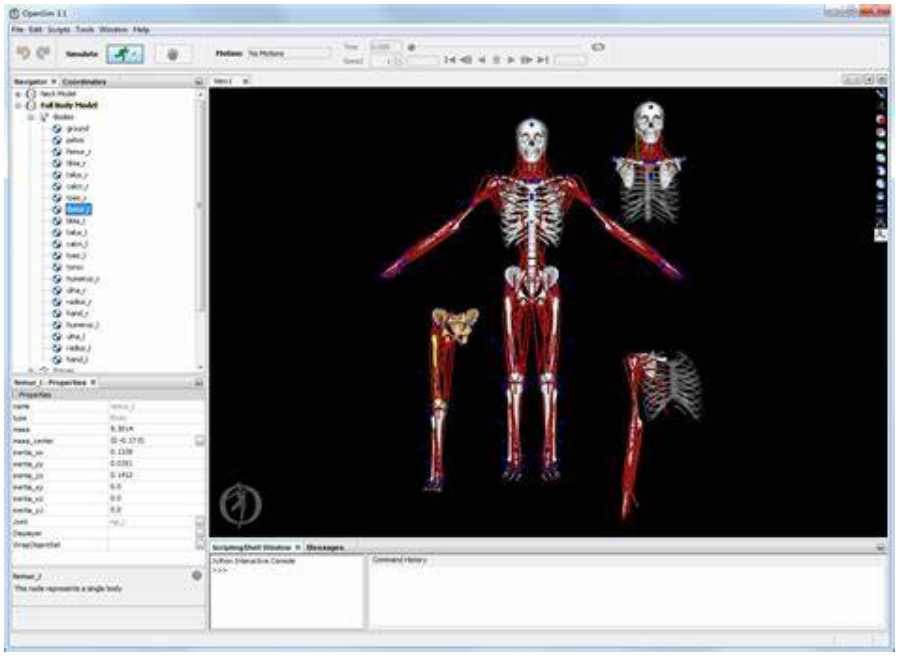


Figure 3.2: OpenSim User Interface.

art software tools and enables others to use these tools in research and clinical applications. It supports a large and growing community of biomechanics and rehabilitation researchers by facilitating the exchange of models and simulations to reproduce and extend discoveries.

The software can be used for any purpose, including non-profit and commercial applications, and the source code is accessible on GitHub, where the community is encouraged to make contributions. OpenSim's platform-specific installers include a GUI and are available at [29].

One of the most interesting features of the program is that starting from a skeletal-muscular model, the user can create its own marker-set thanks to an intuitive drag and drop system.

3.1.1 Marker-Sets

A marker set is the specific arrangement of reflective markers placed on the body of a subject for motion capture [30]. During a static test, the subject remains stationary while the capture system records the position of the markers. The markers are then tracked in space during the dynamic test, and later, during data

analysis, the software defines the position and movement of the subject's body segments based on the set of markers being used. Many different marker sets have been developed and implemented in the state-of-the-art. In some cases, researchers and labs develop marker sets specific to their needs. There is no single model that is considered a universal standard, and the features depend on the type of analysis that is going to be performed.

In this field, OpenSim offers a marker editor that allows access to the currently selected marker in a navigator tool. Markers are used both to scale a generic model to fit a particular subject, and to to resolve coordinate values corresponding to experimentally recorded marker positions thanks to "Inverse Kinematics". Markers can also be used to place reference points on the model. The Marker Editor allows you to change the names of the markers, their offsets, and the frames to which they are attached [28].

In most cases, the model's markers are saved in a separate file, to allow the researchers to use a musculoskeletal model with different sets of markers. The Marker Editor is useful when defining a set of markers for a project for the first time, and for Scale adjusting after the Tool has placed them on a model.

In this work a particular marker set has been used, and it is explained in the next section.

3.2 IOR(Istituto Ortopedico Rizzoli) Marker-Set

As said before, in this work a specific marker set is used, the IOR(Istituto Ortopedico Rizzoli), presented by Leardini et al. [31] and shown in Figure 3.3.

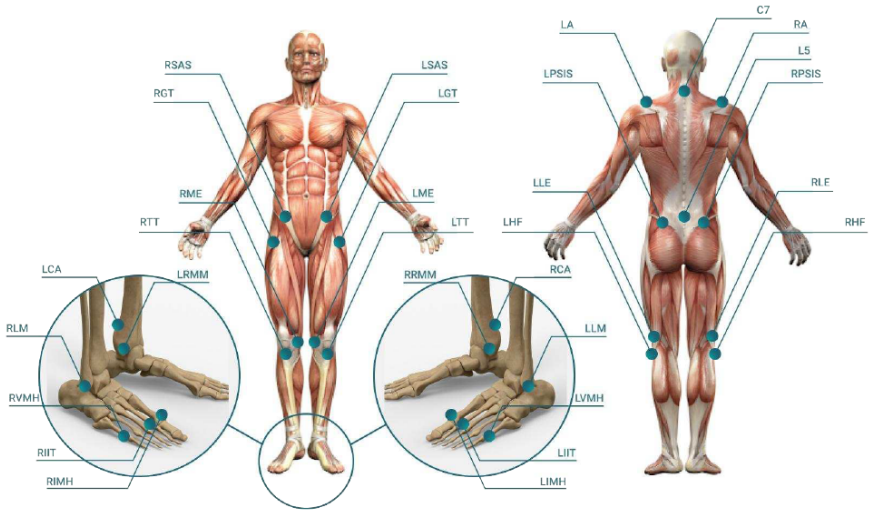


Figure 3.3: IOR Marker-Set.

The IOR markerset is based on the Calibrated Anatomical System Technique (CAST) presented by Cappozzo et al. [32]. The marker-set protocol includes electromyography sensors too as specified by Sawacha, Spolaor et al. [33]. The features are specified in the following table.

Label	Appliance point
Foot	
LCA-RCA	The aspect of Achille's tendon insertion on the calcaneous
LIMH-RIMH	The dorsal margins of the first metatarsal head
LIIT-RIIT	The dorsal margins of the second metatarsal head
LVMH-RVMH	The dorsal margins of the fifth metatarsal head
LMM-RMM	The medial prominence of the medial malleolus
LLM-RLM	The lateral prominence of the lateral malleolus
Knee	
LTT-RTT	The most anterior border of the tibial tuberosity
LHF-RHF	The proximal tip of the head of the fibula
LME-RME	The most medial prominence of the medial epicondyle
LLE-RLE	The most lateral prominence of the lateral epicondyle
Hip	
LGT-RGT	The most lateral prominence of the great trochanter
LASIS-RASIS	The most anterior margin of the anterior iliac spines
LPSIS-RPSIS	The most posterior margin of the posterior iliac spines
Trunk	
L5	The tip of the fifth lumbar spine
LA-RA	The prominence of the acromion
C7	The tip of the vertebra prominens, the seventh cervical line

Figure 3.4: IOR Markers Labels.

The main purpose in using a set of markers with these characteristics, is to give 3D information about the junction points, rather than to identify a body segment. In this way, a software whose goal is any kind of biomechanical analysis, is able to use the joint points directly from the motion capture markers, without having to calculate them, risking errors or inaccuracies.

3.3 Biomechanical 3D File Formats

In this section are presented the main formats used to handle and analyze the data collected from the motion capture, and the software used to open them.

3.3.1 C3D and TDF

The C3D format is a public domain file format that has been used in biomechanics and gait analysis laboratories to record synchronized 3D and analog data since 1980. It is supported by all major manufacturers of 3D motion capture systems and by biomechanics, motion capture and animation companies. C3D files are a standard that contains all the information needed to read, display and analyze 3D motion data with additional analog data(like electromyography, force plates, accelerometers and other sensors) [34].
One of the main feature of c3D is that the format does not change every time a manufacturer releases a new product, so data stored in the C3D format will remain readable for a long time.
The C3D format stores the 3D coordinates and numerical data of any measurement evidence in a single file, removing the need for motion capture data to travel with additional notes and test information.

The second and more common format used over the dataset is the .tdf, which shares some components with the .c3d. This format is returned by Smart Tracker (or SmartAnalyzer) [35], a software used on recording captured by the SMART Motion Capture System showed in section 3.5.8, which has a set of files it uses to store protocols, archives, and regulatory band files. Data saved like this, can be exported to Excel-readable text files and HTML export reports.

3.3.2 Software used to manage 3D files

Mokka

The software used to easily open and check the C3D data is Mokka (Motion Kinematic and Kinetic Analyzer), an open source graphical viewer that helps the

user to visualize 3D data in motion science applications. Mokka is able to support several file formats, including the C3D format, and allows the user to open, import and export data from a wide number of proprietary format producers used in biomechanics [36]. Mokka is designed to view and access 3D data, but does not

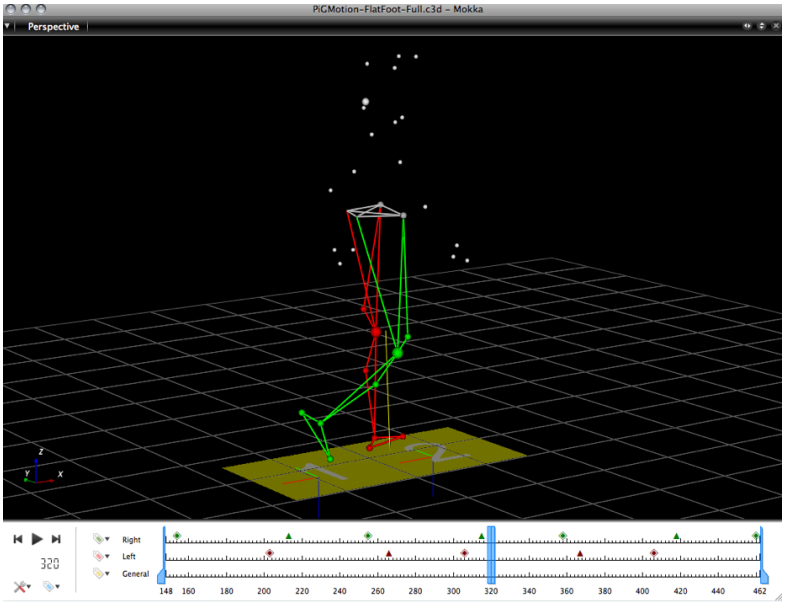


Figure 3.5: Example of Mokka user interface.

provide any editing capabilities for C3D files besides the capacity to remove or add motion capture marker labels.

Smart Tracker

Since Mokka is not able to open correctly heavy .tdf files, another program has been used, Smart Tracker, which makes marker-set analysis more efficient.
The main use of this program, besides the analysis of the dataset made of .tdf format files, can be found in the acquisition phase, where force plates and data from the stereophotogrammetric system (described in section 3.5.1) are processed.
The Smart Tracker is a software of the SMART-SUITE package, which allows to reconstruct three-dimensionally the two-dimensional data acquired with the infrared cameras. The trajectory that executes each marker is assigned a label corresponding to the name of the marker described in the protocol. The main cause of error in trajectory reconstruction is that some markers disappear for short portions of frames or are completely absent.

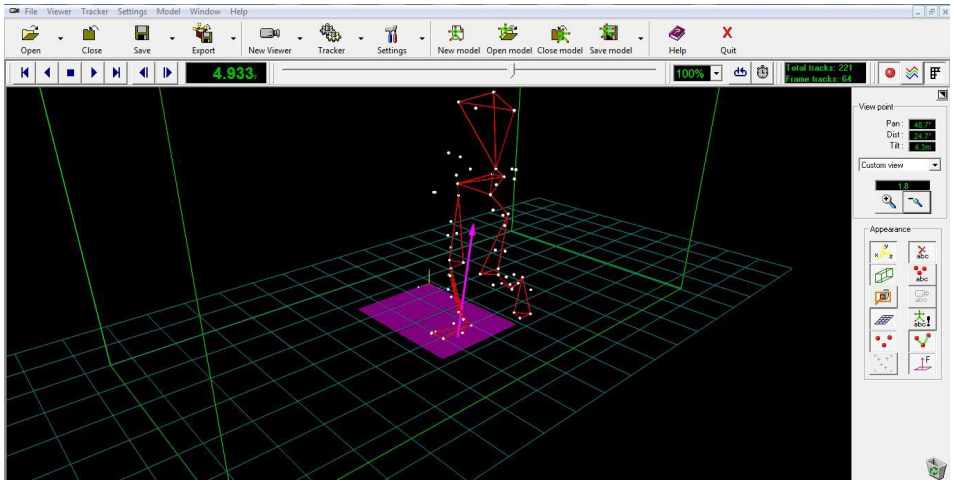


Figure 3.6: Example of Smart Tracker user interface [35].

3.4 BBSOF

The reality of BBSOF was born from university research that over the years has given rise to several collaborations between professionals in biomechanics, high-level sport and medicine.
The company's biomechanical analyses provide an objective and complete description of any sporting gesture. They are therefore intended to support sports professionals (coaches, physical trainers, personal trainers) in the programming of personalized workouts according to the specific motor needs detected in athletes. In the presence of injuries to the lower limbs, especially the knee, the support is also extended to the medical staff (orthopedists, physiatrists, physiotherapists). In this case, the aim is to optimize the rehabilitation process, establish recovery times and ensure the athlete's safe return to competitive activity. Their patented ACL Quick Check® method allows for accurate and highly reliable biomechanical analysis of lower extremity and trunk joints in sport-specific gestures performed by athletes on the training or competition field.
The screening system has very high levels of reliability for primary and secondary prevention of hip, knee and ankle injuries with particular attention to the anterior cruciate ligament. The results of the tests performed with their methods, provide details that are also useful in improving sports performance. ACL Quick Check® is applicable to any single and team sport discipline (rugby, soccer, volleyball, basketball, field hockey, baseball, athletics, golf, skiing, fencing, tennis, dance, skating, iron man...). biomechanical evaluations are performed at the athletes' training/competition site and the quality of the results is comparable to that of the analyses performed in the laboratory [37].

The main fields of interests of the company are:

- **BB-FIELD**: Sport-specific dynamic testing with indoor and outdoor technical apparel at the athletes' site (ACL Quick Check®).
  Instrumentation: Baropodometry - Motion capture 3D - Surface EMG.
- **BB-CLINIC**: Functional tests (gait, squat, single-leg squat, drop jump...), posturography, indoor and outdoor spine mobility at the client's site (ACL Quick Check®).
  Instrumentation: Baropodometry - Motion capture 3D - Surface EMG.
- **BB-Children**:Postural evaluation of walking and running technique. Support and collaboration for the development of targeted exercises and possible insoles.
  Instrumentation: Baropodometry - Motion capture 3D.

3.5 Track On Field

TrackOnField is a software produced directly by BBSoF that allows reconstructing the position of the markers with respect to the position of the absolute reference system, from recordings acquired on field using Go pro cameras, in order to perform a biomechanical analysis on the acquisitions.

3.5.1 Acquisition and Cameras

The acquisition is not part of the software itself, since the real input of Track-On-Field is made by the recorded videos. Anyway, it is the first step of the analysis, and it is worth mentioning.
The acquisition phase is made using commercial GoProHero 7 (Fig. 3.7).

Figure 3.7: GoProHero 7.

The GoPro Hero 7 is a digital camera able to record the subject up to 4K resolution at a rate of 60 fps. These cameras are equipped with a Hyper-Smooth video stabilizer and are much less expensive than the infrared cameras. The GoPro used in the recording sessions are eight and are positioned thanks to the support of a tripod that allows you to adjust the height of the view and ensures excellent stability during filming. Even if the potential is greater, the videos captured with these cameras, according to the needs of the software, have a resolution of 1080p at 30fps.

3.5.2 First session of the software

Opening the software, the user finds in front of him this window (Fig.3.8), where the current session is created and in which the anthropometric data of the subject and the type of protocol used to place markers, have to be set.

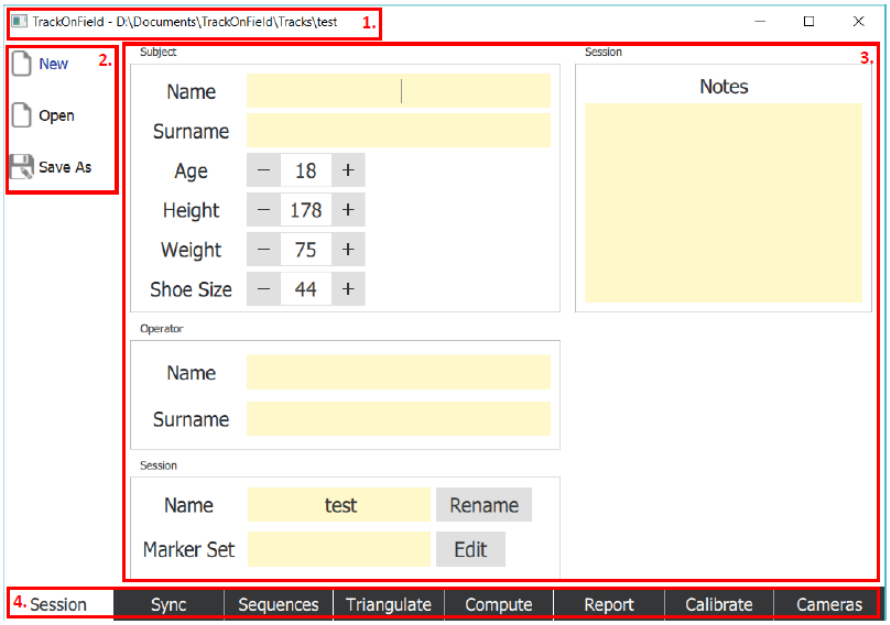


Figure 3.8: Opening session in Track on Field.

The sections demarcated by a red square, correspond to the following features:

1. **Path**: simply the path to the current section.

2. **Load and Save**: options to open an existing profile or generate a new one, and save them.

3. **General identification data and settings of the session**: information about the subject being analyzed in this session, the operator processing the data, and the marker set used.

4. **Tabbed Interface**: each tab represents a module of the application.

3.5.3 Synchronization Phase

In the synchronization phase, after the videos of the acquisitions are loaded, they are synchronized by using the video acquired with camera one (Master) as a reference for all the others.

Figure 3.9: Synchronization panel in Track on Field.

One of the main tools provided by this section is the choice of the frames, to synchronize the videos with the master one specifying frame skews, for each view in case of de-synchronizations that may happen in the different video streams. It's also possible to eject the specific file loaded.

3.5.4 Calibration Phase

In this step the camera is calibrated calculating the intrinsic and extrinsic parameters of it. To calculate the intrinsic parameters it is necessary to load a video in which a chessboard is rotated and the software calculates them using the Bouget algorithm.

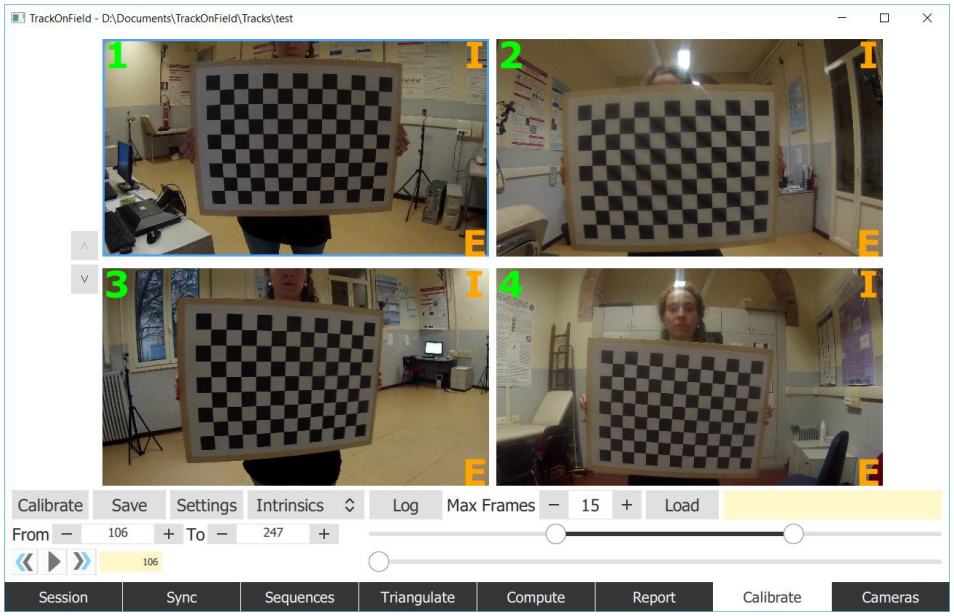


Figure 3.10: Intrinsic calibration.

While for the extrinsic parameters it is necessary first of all to load the video of the acquired data and then manually draw the outline of the rectangle on which the subject was acquired; so that the software builds the absolute reference system placed with the origin on one of the vertices of the rectangle. Once the intrinsic and extrinsic parameters are fixed, the software creates a calibration matrix (exported in .mat format).

3.5.5 Sequences Phase

In this window, the software performs on-screen tracking of markers placed on anatomical landmarks, remembering that all markers must be visible in at least two cameras. Once the respective label is assigned to each marker, the software automatically tracks it following the optical flow frame by frame according to the Kanade, Lucas, Tommasi algorithm [38]. At this stage it is only necessary to check the correct functioning of the automatic tracking and to intervene manually in case the automatic positioning of the marker is not accurate.

3.5.6 Tracking Phase and the need of an auto-labeling implementation

The tracking window allows the software to perform the markers labeling and the marker tracking operations. Generic tracked points are referred to in this window

as features which can be edited.

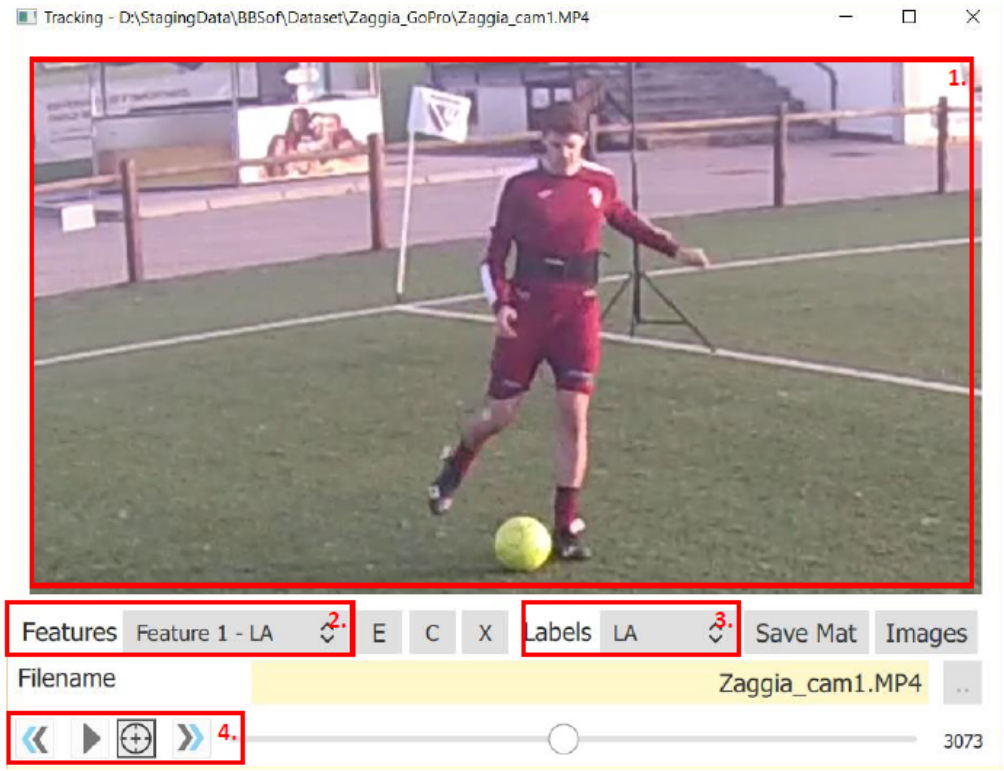


Figure 3.11: Tracking Phase.

Global marker set edit can be opened with "E". Features can be assigned depending on the marker-set. Automatic tracking can be enabled/disabled with the "target" button. It is enabled by default and it works by stepping the frame forward/backward or pressing the play button.

Anyway, this is the main drawback of the software because it requires the lab assistant using the program to manually set the markers first. It is also the starting point for the implementation of the automatic labeling software proposed in this thesis, whose goal is to be able to perform this task, or most of it, automatically.

3.5.7 Triangulate Phase

To proceed with this step the tracking and calibration phase must be completed. After loading the previously obtained calibration matrix and selecting which file has to be triangulated, the software builds a matrix in which the instantaneous position of the markers in 3D space with respect to the absolute reference system is inserted.

Triangulation involves reconstructing the position of the markers in space (x, y, z) from the coordinates of their projections in the image plane reference systems of the cameras (x, y). At least two 2D points are required for 3D reconstruction.
Once the 3D position of each marker has been reconstructed for each instant of time, its trajectories have to be calculated.

This Phase returns an output with the same features of the output of the auto-labeling algorithm implemented, that is a cloud of 3D positions labeled following the labels of the marker-set.

3.5.8 Direct acquisitions using infrared and Smart Tracker

Another kind of acquisition, that consents to avoid the process performed by Track on Field preprocessing and triangularization, is performed using stereophotogrammetric system cameras. Differently from the Gopro recordings, the stereophotogrammetric system consists of six infrared cameras shown in figure 3.12. These

Figure 3.12: SMART DX infrared cameras.

cameras are manufactured by BTS Bioengineering and are part of the SMART E line. These instruments are used worldwide for clinical assessment and multi-factorial motion analysis [39]. They feature extreme computational power and exceptional versatility; they also employ powerful infrared illuminators to ensure excellent performance even in low-light conditions. The sampling rate used is 60 Hz, extendable up to 120Hz.
Thanks to these particular cameras, and the software Smart Tracker presented in section 3.3.2, it is possible to avoid most of the functions required by Track on Field for the acquisitions editing.
Besides the convenience of using this system to avoid the phases described for

Track on Field, this software can be used in a controlled environment, and is not suitable for a sport outdoor field. Even with acquisition performed with Smart infrared cameras, the tracking phase is necessary, but can be performed directly on Smart Tracker.

3.5.9 Matlab Analysis

At this point, the labeled 3D acquisition, retrieved with track on field or using the Smart Tracker, is ready to be analyzed.
The compute module is featured for processing and report producing. Multi-protocols are available, and the compute module supports the creation of trials made by collections of labeled sequences. Each trial can be renamed, and once saved it is available to the Matlab interface. Furthermore, screen bottoms are implemented to compute, debug, or to open the directory with the outputs.
Similarly to an execute protocol aggregating trials created during the compute module, a report module is introduced in the matlab interfacing. By selecting the trial, it can be processed with Matlab, the possible outputs based on the needing, that the user is able to obtain with this software are the following:

- Bands of normality of joint angles of flexion/extension, ab/abduction and intra/extra rotation of hip, knee, ankle, pelvis and trunk both with stereophotogrammetry data and with video analysis data.
- Normal ranges of joint moments of flexion/extension and ad-abduction of hip, knee and ankle calculated with the "Ground Reaction Vector technique" both with data from stereophotogrammetry and with data from video analysis.
- Bands of normality of plantar pressures.

4 Neural Network Tools

4.1 Recurrent Neural Networks

In natural environments, each sensory state is strongly correlated with previous ones. Our brains are constantly engaged in predictive processing, where past information is used to guide current sensory processing. This allows contextual information to be considered in the temporal dimension during inference and learning (top-down processing), and also allows plausible sensory information to be generated even when there is no stimulation. All this is made possible by the presence of feedback connections [40].
The basic idea behind a Recurrent neural network is to exploit a set of time delayed, feedback connections to store past information in the hidden layer. In this way the input is no more a static vector, but rather a time series of ordered vectors, and the model is able to behave like a dynamical system.

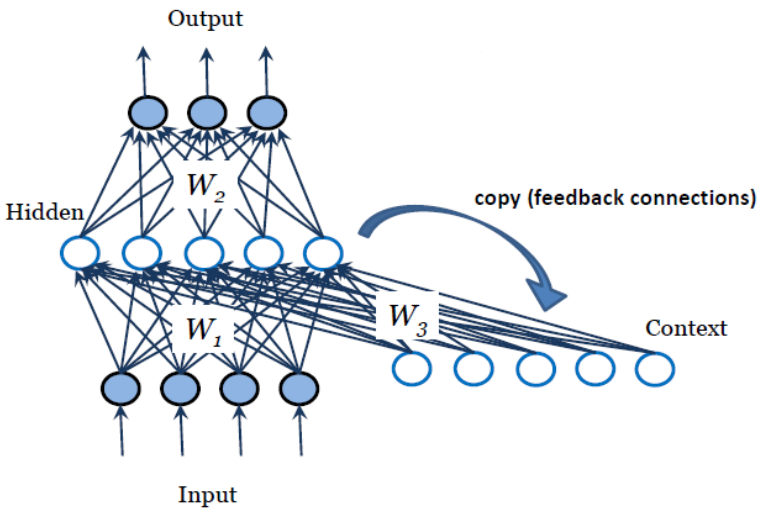


Figure 4.1: Recurrent Networks Architecture (1).

From a technical point of view, for each input of the network, the activation of hidden units is copied to the context layer, and the activation at the next time step is influenced by the state of the context layer. In this way the Network predicts the next input rather than simply producing an output.

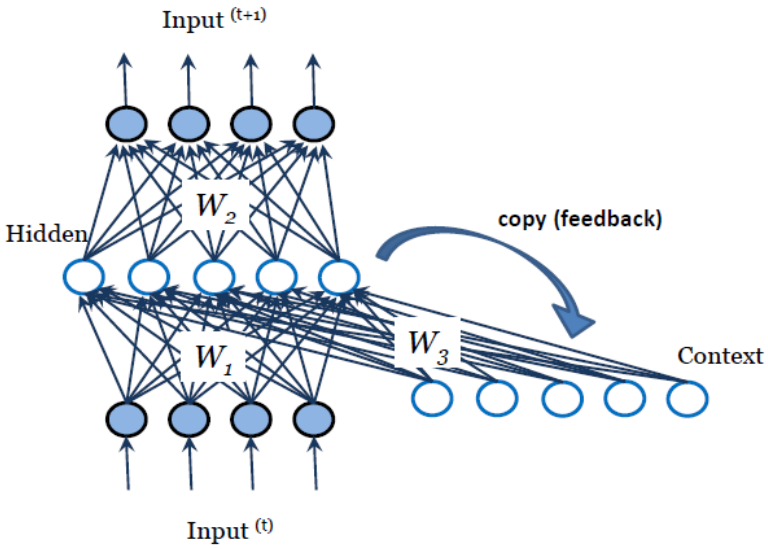


Figure 4.2: Recurrent Networks Architecture (2).

One of the main uses of this architecture is to find structure in time, as the work proposed from Elman et al. [40] in which the network learns how to predict the next element in a sequence of English sentences. The work shows how the network can discover lexical categories, clustering the words according to their statistical usage (i.e., based on the surrounding context), discovering several major categories.

A recurrent network can thus be seen as a very deep network with parameter sharing, in which rather than computing the gradients only at the end of the unrolling, we can break down the sequence into shorter pieces (truncated BPTT). It is interesting to consider how the unrolling of a computational graph over the time, can transform the recurrent architecture into a directed acyclic graph [41].

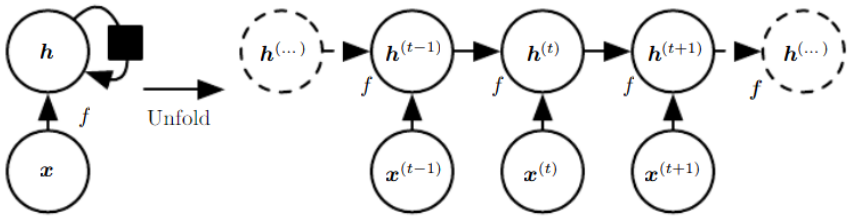


Figure 4.3: Directed Acyclic Graph from RNN.

There is a variety of sequence learning regimens:

- **Sequence to sequence**: each input is followed by a prediction.

- **Encoder decoder**: sequence to sequence model where the entire input sequence is first encoded into a fixed length vector, and then the entire output sequence is generated.

- **Sequence to output**: one single output is produced, at the end of the sequence.

4.1.1 Universal Approximation Properties

Over the years several theorems have been introduced to support the state-of-the-art of Recurrent Neural Networks, and these are called "Universal Approximation Properties":

- For any computable function, there exists a finite recurrent neural network that can compute it [Siegelmann and Sontag, 1992] [42].

- Any finite time trajectory of a given n dimensional dynamical system can be approximately realized by the state of the output units of a continuous time recurrent neural network with n output units, some hidden units, and an appropriate initial condition [Funahashi and Nakamura, 1993] [43].

- Any open dynamical system can be approximated with an arbitrary accuracy by a recurrent neural network defined in state space model form [Schafer and Zimmermann,2006] [44].

4.2 Long Short Memory Term Networks (LSTM)

A Vanilla Recurrent Neural Network can effectively learn short term temporal dependencies, however, learning long term dependencies may be hard due to gradient vanishing.
RNNs are trained through BPTT (Backpropagation through time), so the weights of the neural network get updated through the gradients. But this backpropagation is performed through time and the layers, and with a large sequence, the gradients may become smaller and smaller over the backpropagation, and it could even vanish.
Another drawback of RNNs is related to the long term dependencies, a common issue in language models. RNNs have been found to handle long-term dependencies poorly, and performance decreases as the gap between the event and the context increases. A clear example of this can be found in a long sequence of sentences 4.4.

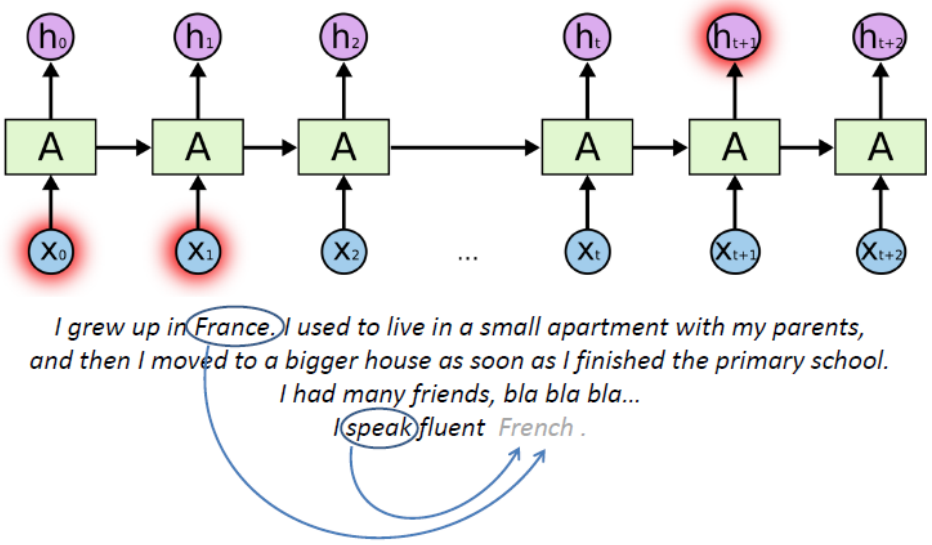


Figure 4.4: Long term dependencies in a language model.

To solve this problems, we can add to the vanilla RNN, a set of gated units able to learn how much information should be retained in temporary memory, and how forward it should be propagated.
In this way a Long Short Memory Term Network (LSTM) is introduced, a network specifically designed to overcome the disadvantages of RNNs, able to preserve information for longer periods, and to solve the vanishing of the gradient too.

In a Recurrent Neural Network, we usually have a network layer repeated for t time steps. Within this module, RNN has a neural network with a simple activation layer, usually a tanh or a sigmoid, but other similar functions are available too (see Fig 4.5).

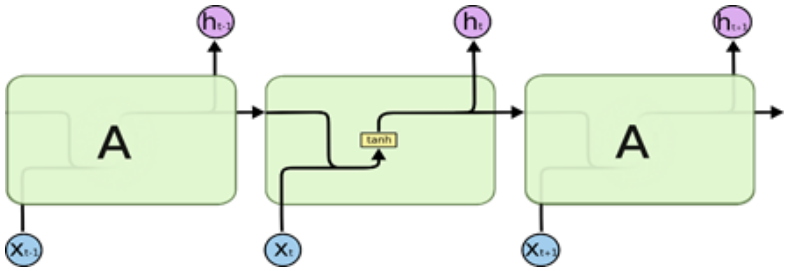


Figure 4.5: RNN network units.

To evaluate in a proper way all the long-term and short -term events analyzed, it is necessary to add a long-term memory and a short-term memory, in order to provide the right information, and discard the others.

4.2.1 LSTM Architecture

While a RNN unit has a single input and a single output (Fig. 4.5), the LSTM unit has two different inputs and two different outputs (Fig. 4.6).

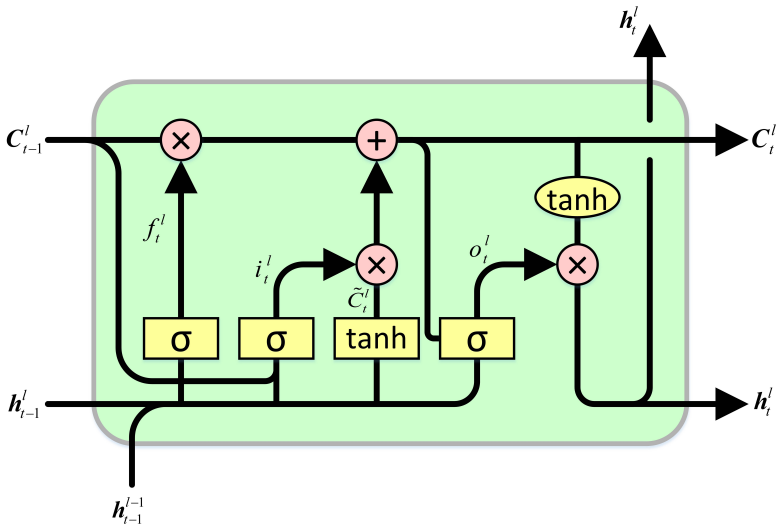
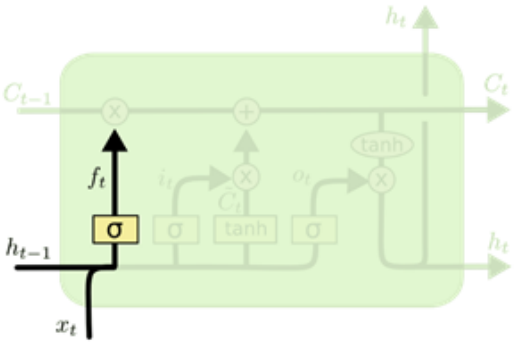


Figure 4.6: LSTM network unit [45].

The hidden state *ht* is the short-term memory that comes as input from the previous unit. The additional parameter that can be found is the vector Ct, known as the cell state, and responsible to store long-term memory events.
In order to add or remove information from the units, gate mechanisms are used:

Forget Gate



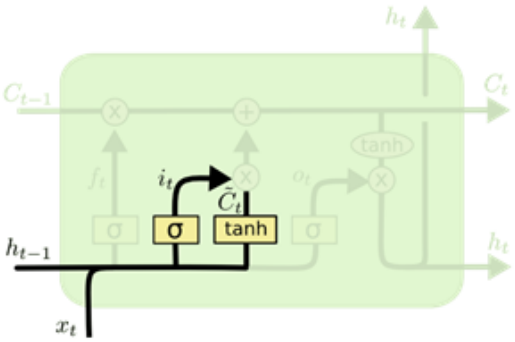
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 4.7: LSTM Forget Gate.

The cell state or the long-term memory from the previous time step is multiplied with a function ft to obtain the new filtered memory, where ft is the forgetting factor. The forgetting factor is calculated using the formula shown in Fig. 4.7.
The short-term memory (ht from the previous timestamp) and the current event are used to calculate the forgetting factor. The short-term memory and the current event are concatenated, and a sigmoid layer is applied on this vector. The sigmoid function produces an output from 0 to 1 which is then multiplied with each value in the previous state of the cell.
A value of 1 indicates that the information is kept as it was, while a value of 0 indicates that the information will be completely discarded.

Learn Gate

The learn gate is responsible to find the new information needed to be added, once the previous one has been discarded or not.



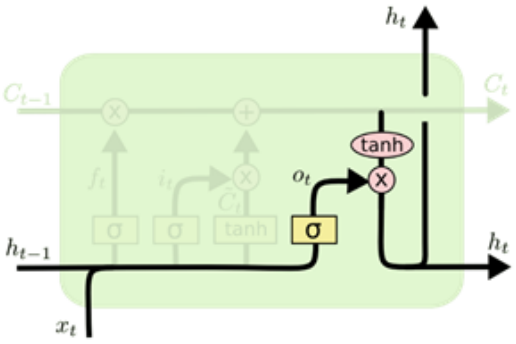
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 4.8: LSTM Learn Gate.

The previous short-term memory and the current event are concatenated and then passed through a *tanh* layer, which generates new values representing the new information.
Another forget gate determines how much of the new information gets updated, and the output from the forget get is multiplied with the new information in order to generate the final output.
The activation function $Tanh$ is able to output vectors (in the range -1 to 1) that are centered around 0, to distribute the gradients, allowing the cell states to run longer. In this way it is possible to avoid the vanishing or the exploding of the gradients (Fig. 4.8).

Remember and Output Gate

In the remember gate the cell state or the long-term memory are updated adding the output of the learning gate to the forget gate one.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 4.9: LSTM Output Gate.

The final part of the cell is the output gate, in which we retrieve an output that represents a filtered version of the cell state. The value of the cell state from the remember gate is multiplied by an activation function *tanh*, and then is multiplied again through a *sigmoid* function, to drop some of the values (Fig. 4.9) [45].

5 Auto Labeling Algorithm Implemented

This chapter is about the technical side of the Algorithm implemented to perform the automatic labeling of the motion capture markers, starting from the framework used and its different libraries. It will follow with the analysis of the dataset, that requires a particular attention and work itself. Last but not least, the Algorithm itself will be explained in detail.

5.1 Python for Artificial Intelligence

Python is one of the most popular programming languages used by developers until today. It was created by Guido Van Rossum in 1991 and since then it has been one of the most widely used languages along with C++, Java, etc. Python is an "interpreted" language which does not need to be compiled into machine language instructions before execution and can be used by the developer directly to run the program. This makes it complete enough to be interpreted by an emulator or virtual machine on top of the native machine language [46].

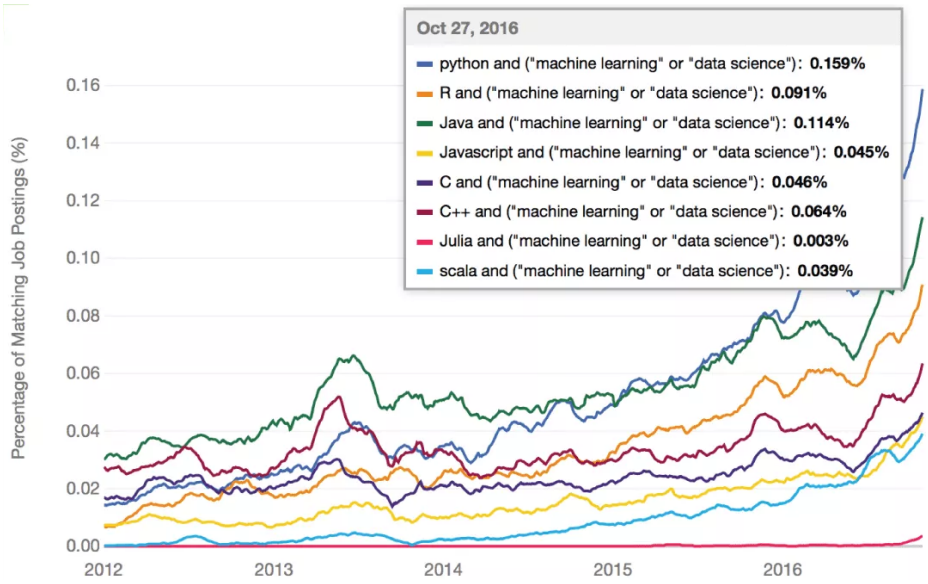


Figure 5.1: Python as the best programming language for AI and ML [47].

One of the reasons for its success in the world of Artificial Intelligence, is the fact that it has pre-built libraries for scientific computation, advanced computing and for machine learning. It is also easy to find comprehensive support and assistance thanks to forums and tutorials, making the job of the coder easier than any other languages as c++ and Java.
Furthermore, the flexibility of the framework allows developers to choose the programming styles they are more comfortable with or even combine different styles to solve many types of problems in the most efficient way. It is also platform independent, requiring little changes in the case of moving between Windows, Linux, Macos, Unix, and others. There are many other factors that influence the success of Python, that for these reasons has been the framework used to implement the algorithm that will be described in this chapter, starting from the main libraries, presented in the next section.

5.1.1 Python's Main Libraries Used

Only the main libraries will be reported here, to explain the choice of them and how they are used through the Algorithm.

- **Pytorch**: is an open source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, developed primarily by Facebook's AI Research Lab (FAIR). It is free and open source software released under the modified BSD license.
  It is the framework used to handle every part of the algorithm connected with the Neural Network, from the tensor in input, to the data-loader and the network itself.

- **Numpy**: is an open source library for the Python programming language that adds support for large matrices and multidimensional arrays along with a large collection of high-level mathematical functions to efficiently operate on these data structures. NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter, speeding up mathematical operations by providing multidimensional arrays, functions and operators.

- **EZC3D**: a toolkit able to read, modify and write .C3D format files. It is written in C++ with appropriate binders for the Python and MATLAB/Octave scripting languages [48].

- **BTK (Biomechanical Toolkit)**: similarly to EZC3d, BTK is an open-source, cross-platform library for biomechanical analysis. It is able to read, modify and write .tdf format files. Even if the toolkit is again native in c++

API, it includes adaptations for Matlab and Python.
The core of BTK is based on a pipelined design and shared pointers. Each process can be linked and scheduled together, and the use of shared pointers avoids the need for memory allocation and deletion, the choice of object owner, and the possibility of memory leaks [49].

- **Ray Tune (from Ray)**: tool used to perform the hyper-parameters tuning explained in section 5.3.8, easily distribute trial runs to quickly find the best hyper-parameters, automatically discarding cases of gradient descent, slow learning or other problems connected with a bad learning of Neural Networks.

5.2 Dataset

5.2.1 Motor Task Analysis

The Analysis of the dataset has been a time consuming process. Firstly because the 3D file had to be opened with the relative tools (that were different for C3D and TDF) one by one, to associate the correct motor task. This was necessary in order to consider other sub-analysis in which the network is trained and tested on a specific task. In particular the movements in analysis were the following:

- **Static**: is the first acquisition performed, in which the subject does not move. Anyway there are micro-movements due to breathing and balance adjustment. Thanks to the biomechanical analysis, it is already possible to detect postural problems during the analysis of a static.

- **Gait**: a simple back and forth walk. The analysis of gate is an essential part of diagnosing various neurological disorders and evaluating a patient's progress during rehabilitation and recovery from the effects of neurological disease, musculoskeletal injury or disease process, or lower extremity amputation. Furthermore, it is a way to check whether the subject has correct posture.

- **Hip rotation**: internal hip rotation activates the muscles of the hip, buttocks, and thighs. An insufficient hip internal rotation can lead to gait problems. For example, the knees or soles of the feet may sag inward. When other parts of the lower body compensate for poor hip internal rotation, the risk of an injury may increase.

- **Squat**: performed with the left leg, the right leg, or both. The squat is a strength exercise in which the subject lowers the hips from a standing position and then stands up. During the descent of a squat, the hip and knee joints flex while the ankle joint dorsiflexion; conversely, the hip and knee joints

extend and the ankle joint plants when standing up.
Squats are considered a vital exercise for increasing the strength and size of lower body muscles and developing core strength. The main agonist muscles used during the squat are the quadriceps femoris, adductor major and gluteus maximus. The squat also uses the erector spinae and abdominal muscles isometrically, among others.

- **Drop**: performed with the left leg, the right leg, or both. It is an action whereby the subject jumps from a raised platform and lands trying to break the fall.

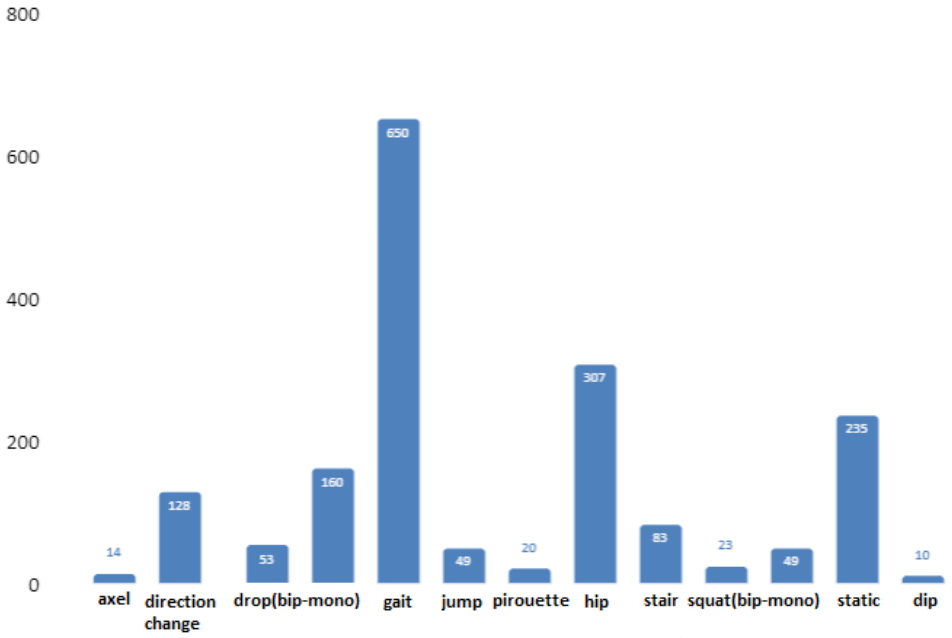


Figure 5.2: Number of files per Motor Task.

- **Step**: the subject simply climbs one or more steps.
- **Jump**: the subject simply jumps and lands.
- **Axel**: skating related movement, the thrust to execute the jump is given by the extension of the supporting leg, helped by the momentum of the free leg, while the rotation is given by the starting wire (external) and by the fast recall of the right arm and leg. In the air, one and a half rotations are performed and you land on the right wire back outside. During the flight phase, the arms and legs must be kept as close as possible to the axis of rotation of the body. The approach of the masses (arms and legs) to the

rotation axis involves a decrease in the moment of inertia and therefore an increase in angular velocity since the two entities are inversely proportional to each other.

- **Pirouette**: skating related movement, figure consisting of a turn of the body pivoting on one leg only.

5.2.2 3D files Analysis

Besides the analysis of the movements, it was necessary to perform an analysis of the 3D files that are given in input to the network. In particular most of the acquisition were not acquired in a controlled environment, but mostly on the field. Because of this, the 3D files are noisy, with the presence of missing markers due to occlusions, and of extraneous markers, wrongly acquired by the environment itself. In other cases, the recording started before the subject entered in the field of recording, returning a file that is mostly empty, or it lasted too long, providing the same problem. For this reasons, besides the software implemented for the automatic labeling, it has been necessary to implement another software, devoted only to the analysis of the dataset.
Since every file was in specific positions inside the database of the company, the first step has been the generation of an .xml data-frame containing in each row the address of a specific C3D or TDF file. Than for each one the motor task has been manually written in the data-frame.
From this .xml file, the new software developed is able to connect directly to the company's Network Attached Storage and to open each file in order to retrieve the following information:

- **Extension**: returns the extension of the file (C3D or TDF).
- **Total number of frame**: to understand the length of the 3D file, being able to discard data that are too short.
- **Number of frames with at least one marker**: to confront the total number of markers, and avoid using files that are mostly empty.
- **Mean number of markers**: to confront the max number of markers to understand if there are un-labeled markers in the 3D file.
- **Max number of markers**: to check if there are less or more markers compared to the dataframe.

This process returns a dataframe (.xml) with the path of a single c3D or TDF file, to directly load it from the NAS, followed by the features named above, being able

to filter files to take on precise needing.

The ideal 3D file has the same mean and max number of markers, that coincides with the marker-set used. However, 3D files like these are a minority in the dataset, and for this reason the auto-Labeling algorithm implemented has to take actions accordingly to this.

5.3 Auto-Labeling Algorithm Implemented

5.3.1 Import of the 3D files

The first step in the algorithm is to import the files (C3D and TDF) in the proper way. Since these 3D structures are not the most common file as could be a set of images or some numerical values in a data-frame, the program requires proper libraries and associated functions to extract the 3D coordinates into a 3D Python structure (Numpy's ndarrays and tensors).
For this, two sets of parallel functions have been defined, one for the .c3d files and the other one for the .tdf files. These functions aim to load the 3D file, then get the 3D coordinates of the m-th marker at the i-th frame and return all the labels of the 3D file too. The next step is to generate a Numpy ndarray that contains the 3D coordinates of the 3D file for all markers, at all frames available. Furthermore two functions are used to get an array with the number of visible markers at the each frame, and the number of frames with at least one marker tracked (as an int). Once all the data described is ready, a function uses two of the functions described above to return a Numpy's ndarray with the following shape: number of frames $\times$ number of markers $\times$ three-dimensional axis [x,y,z] (Fig. 5.3).

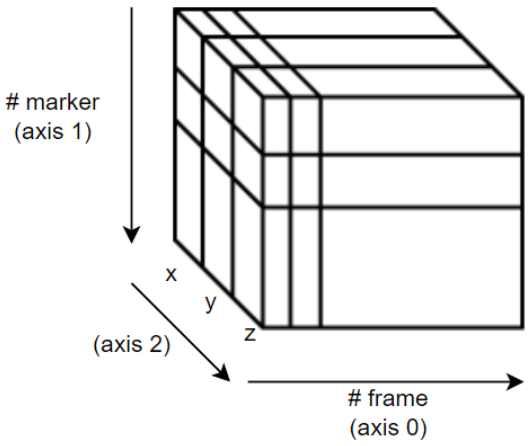


Figure 5.3: Numpy's ndarray containing the 3D data.

In this way the program is able to generate one of these structure of data for each 3D files it receives in input.
All these files are then interpolated with a linear interpolation on gaps values under a specific value. Next, a butter-worth low pass filter is applied along frames of a single coordinate for all markers to make the oscillations less abrupt.
The 3D files are so transformed into tensors as needed from later interactions, and each file get reordered basing on the marker-set to keep trace of the marker in analysis without the needing of a labeling value inside the cells.
If the shape of the markers is smaller than the number of markers reported in the .xml file of the marker-set, horizontal planes ([x,y,z] $\times$ n.frame) are generated with null values(NaN).
Once the single 3D tensors are ready, they are added inside a list (Fig. 5.4), generating the input of the pre-processing part that prepares those files to become the input of the Neural Network.

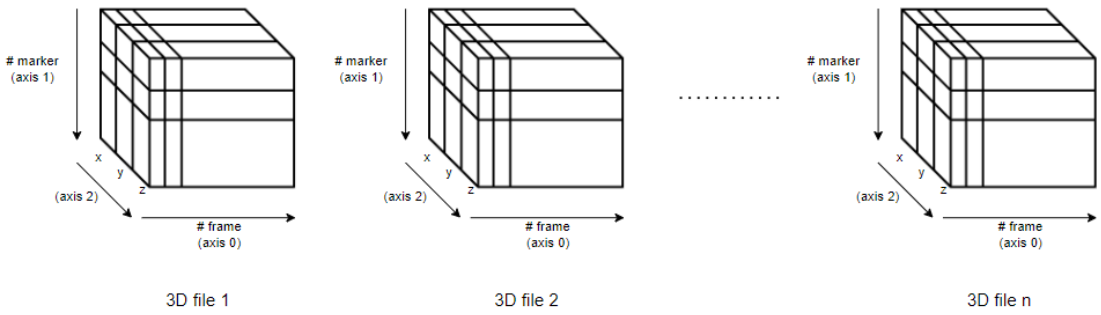


Figure 5.4: List of reordered and filtered tensors.

5.3.2 Windows to Segment Data

The next step is to define time windows that consist of a list containing:

- the relative C3D file.
- the number of the C3D marker.
- the starting frame.
- the arrival frame.

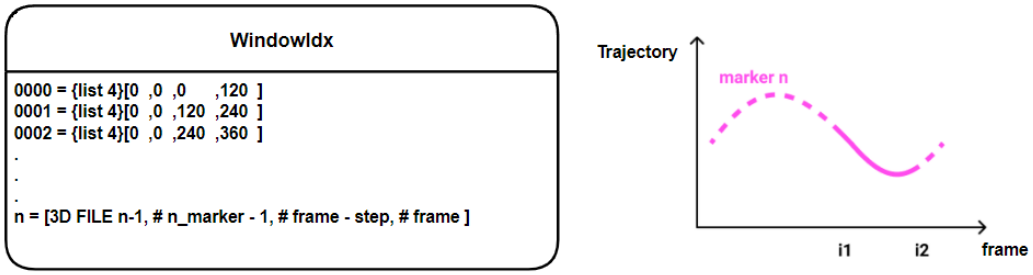


Figure 5.5: Windows structure with a step size of 120.

Those windows are used later to prepare the instances of input for the Neural Network, and to be generated they need as an input the data list, the number of markers and a chosen step size that can be modified during the parameters tuning (Fig. 5.8).

5.3.3 Pre-processing Function

The preprocessing function is used on both the input of the neural network and the prediction performed by it. Once the list of tensors containing the 3D files and the list of windows to segment them is ready, the class to prepare the input of the LSTM Network is introduced using the data primitive torch.utils.data.Dataset provided by Pytorch.
The "_init_" function initialize the values of the class itself, the number of markers, the windows, and the values of scaling (not used in the inter-distances version).
The "_len_" function returns the number of items in the dataset.
Lastly but not for importance the function "_getitem_" performs the preparation for the input itself. Here, after the values of the idx-th element of windowIdx = [0, m, i1, i2] are taken, it performs a slice between i1 and i2, so it takes the time step depending on the size of the window defined. These slice values are isolated relatively to the m-th marker taken into consideration.

3D coordinates interpolation

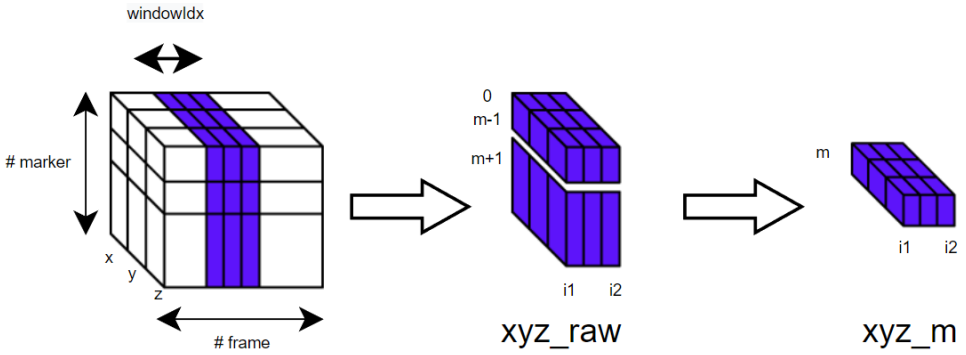


Figure 5.6: 3D coordinates interpolation.

- If there are no gaps in the xyz_raw tensor, it calculates the distance between xyz_raw (the raw tensor) and xyz_m (the slice of tensor relative to the marker m considered), performing a point-wise difference (which corresponds to a discrete derivative), and calculate the norm (Euclidean distance in axis 2 direction, i.e. calculate how far it is from 0, ideally the norm would be 0 if the marker in question had the same m-th marker position at each frame). Now it calculates the average in axis 0 direction and does an ascending sort. The topmost marker is the one with the average trajectory most similar to the m-th marker.

- Otherwise, markers with less missing values are sorted in descending sense.

If xyz_raw has some missing value, the values in the cell (i*, m*, c*)-th are replaced by the average values of the c* coordinate (i.e. x, y or z) of all markers on the i* frame.
Markers are then ordered by putting those with the most similar trajectories (i.e., the average of the frame-by-frame position differences of the 3D coordinates) at the m-th marker with the highest value.

Speed calculation

The value of the speed for each marker is calculated with a pointwise difference (i.e. 3D translation tensor).

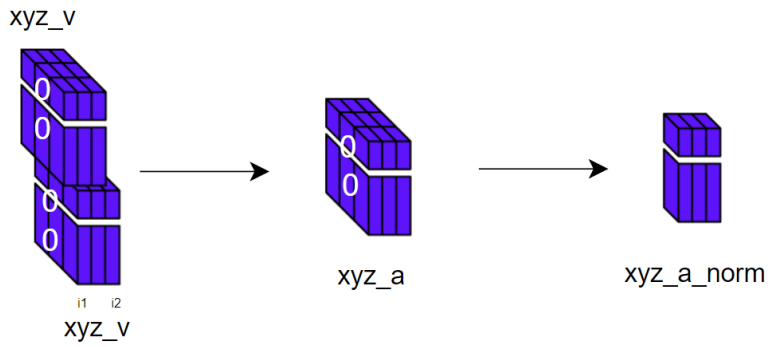


Figure 5.7: Speeds calculated with tensor translation.

Then the norm is calculated along the second axis (as for the 3D coordinates interpolation seen above) of the translation tensor. So we can see this "velocity matrix" as an "intensity matrix of frame by frame translations".

Acceleration calculation

The value of the acceleration for each marker is calculated with another pointwise difference, but this time it is performed on the speed values.

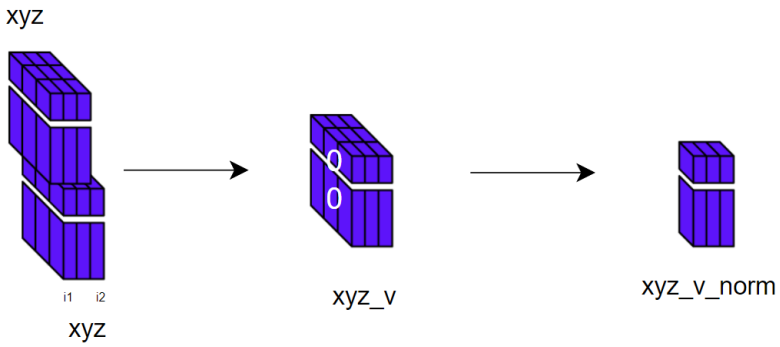


Figure 5.8: Accelerations calculated with tensor translation.

Then the norm is again calculated along the second axis of the translation tensor.

Scaling and Stacking

The tensors of positions (xyz), speed (xyz_v_norm), and acceleration (xyz_a_norm), are than scaled with using the scaling values described in the next section (5.3.4). Furthermore, a stacking is performed between xyz, xyz_v_norm and xyz_a_norm along axis 2.

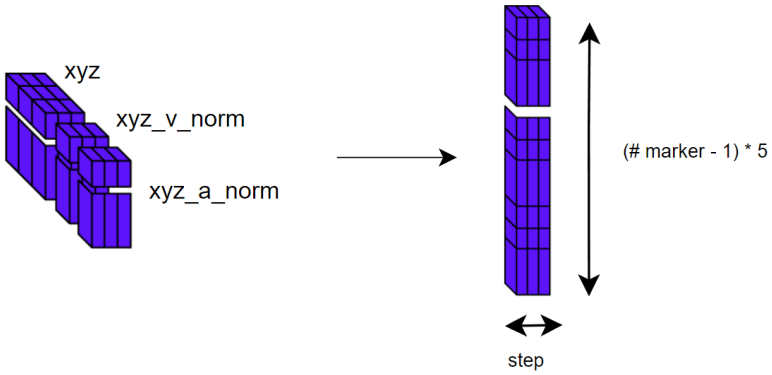


Figure 5.9: Stacking of positions, velocities and accelerations.

Finally the input is ready, and the function "_getitem_" returns the stacked 3D structure and the marker and the window taken into consideration.

5.3.4 Scaling Values

With the same procedure as seen in the previous section, the Algorithm uses tensors of positions, distances and accelerations to generate the values used to scale the input of the Neural Networks. The Algorithm proceeds as it follow:

1. Sums the absolute values of the inter-distances obtained from all markers of all 3D files.

2. Sums the absolute values of the velocities (normalized) obtained from all the markers of all the 3D files.

3. Sums of the absolute values of the accelerations (normalized) obtained from all markers of all 3D files.

4. Counts the number of distances taken into consideration.

5. Counts the number of (normalized) velocities taken into account.

6. Counts the number of (normalized) accelerations.

At this point calculates the averages of the inter-distances, speeds and accelerations calculated in the previous point. These are the actual scale values that will be used both in the training and in the prediction phase.

5.3.5 Inter-distances alternative

Another version of the same algorithm built, makes use of the inter-distances instead of the normal distances from the origin of the axis. This version has been introduced to analyze in a more effective way those files that have not been collected in a controlled environment, allowing to ignore the dependence on the origin of the axes. The process is the same as the one seen in section 5.3.3, differing only for 3D coordinate interpolation, that is performed on the inter-distances instead. Velocities and accelerations are then calculated in the same way. The second difference is that the scaling values seen in section 5.3.4 are less relevant to apply to the values found because of their nature, so in the inter-distances alternative they are not calculated or necessary at all.
This version, even if added in a second moment, has resulted to be more suitable to the dataset in analysis, and it has been used to get the results.
The remaining part of the algorithm works in the same way, so the two versions are easy to exchange as needed.

5.3.6 Neural Network architecture

The Neural Network used is a Long Short Term Memory Network explained in chapter 4.2.1, and it is used to calculate the probabilities of the labels for each window.

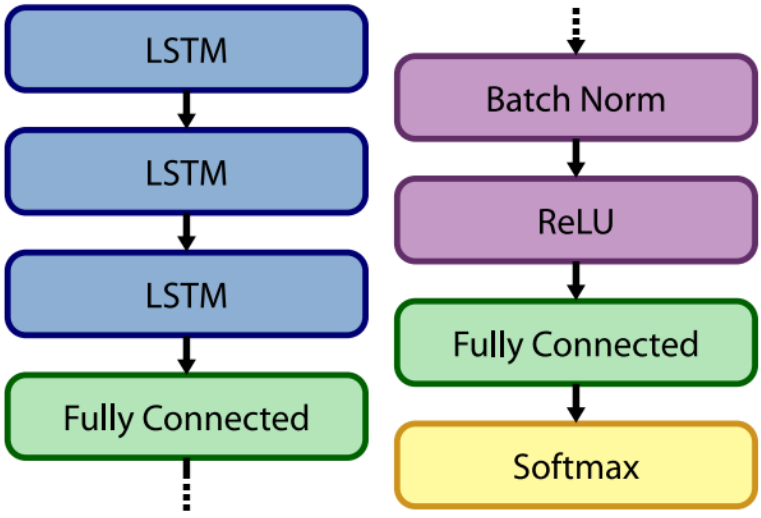


Figure 5.10: Simplified Architecture of the LSTM used.

The network is made firstly by three recurrent LSTM layers in which is applied a dropout variable from the tuning of hyper-parameters. These are followed by a

Fully connected layer, and a one dimensional batch normalization, a method used to make artificial Neural Networks faster and more stable by normalizing layer inputs through re-centering and scaling, proposed by Sergey Ioffe and Christian Szegedy in 2015 [50]. After the BatchNorm there is a ReLU (rectified linear unit), a nonlinear function (or piecewise linear function) that will output the input directly if it is positive, otherwise, it will output zero. ReLU is the most commonly used activation function in neural networks. The last part is a softmax function, a generalization of the multinomial logistic function usually used as the last activation function of a neural network to normalize the output of a network to a probability distribution over the predicted output classes (as in this case). The Network is generated starting from the PyTorch class torch.nn.Module, and the forward function that receives in input the dataset and the dataset length is built as it follows:

1. out = torch.nn.utils.rnn.pack_padded_sequence(data, data_lens): Generate the input for the LSTM (generate a Packed Sequence)
2. out = LSTM(out): Generate the prediction through the LSTM (output is still a Packed Sequence)
3. out = torch.nn.utils.rnn.pad_packed_sequence(out, data_lens): Converts the output to a sequences with padding (as the initial one)
4. out = FullyConnectedNN(out)
5. out = Tensor(batch size x number of markers, figure 5.11)

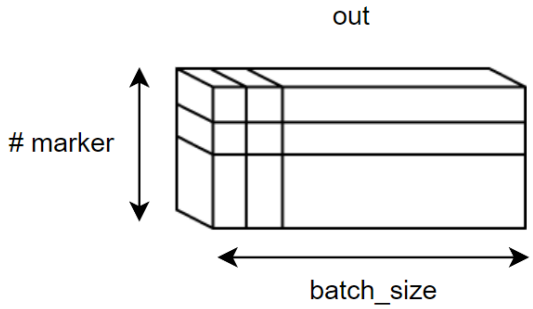


Figure 5.11: Output of the LSTM.

5.3.7 Collate function used to adapt the batch

Before speaking about the training features, it is important to point out that the data-loader batch has been modified with an additional function, to make it more

suitable for the data structure. In particular this function zips into 3 separate batch lists the data provided by the pre-processing function, the markers, and the window considered. Next it adds 0-padding along axis 0 to each element of the input data, so that all elements have the same size. Then it converts the input data described in chapter 5.3.3 and shown in figure 5.9 (X) into a tensor by stacking along axis 0 the elements long as the size of the batch selected, returning X_pad (Fig. 5.12).

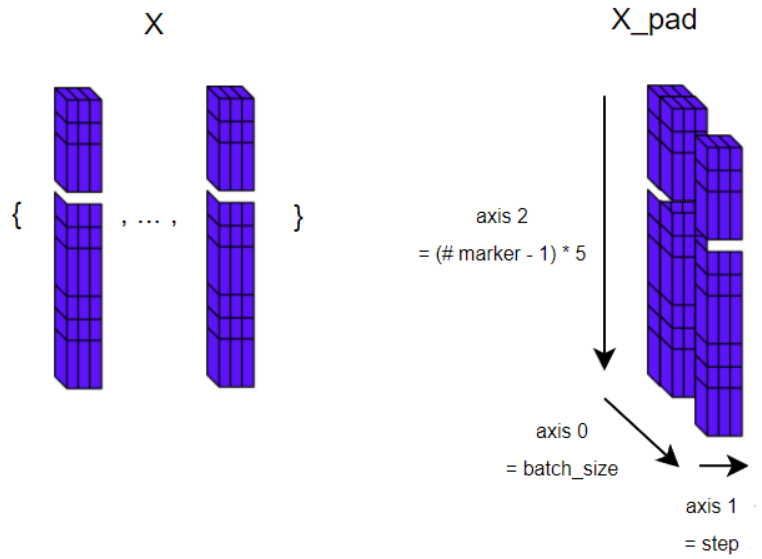


Figure 5.12: Collate function output.

5.3.8 Hyper-parameter tuning

The hyper-parameter tuning has been performed to chose the number of layers of LSTMs, the number of cells of each of these layers, the number of nodes of the fully connected layer, the dropout percentage, the learning rate, the momentum and the batch size.
The tuning has been performed with the help of Ray Tune, an industry standard tool for distributed hyper-parameter tuning, that includes the latest hyper-parameter search algorithms, integrates with Tensor-Board and other analysis libraries, and that supports distributed training through Ray's distributed machine learning engine [51].
Data loaders are wrapped in their own function and pass a global data directory. In this way a data directory can be shared between different processes.
Only those parameters that are configurable can be tuned. The configuration parameter will receive the hyper-parameters to train to train the network, and a

function of checkpoint is used to take pauses if needed and restore checkpoints. After the algorithm has divided the dataset in training, validation and test set, it performs a comparison at many iterations using the range of hyper-parameters given as "configuration" parameters. Then it returns the best combination it has found.

5.3.9 Training features

To perform the training, the algorithm firstly initializes the pre-processing function and the data-loader, to prepare the proper input for the network. Calculate highest length by taking maximum shape[0] from the dataset list of elements (i.e. the number of frames of the element with the maximum number of frames). This value is used as a parameter to initialize the network. Than the network is initialized on the tuned parameters and the maximal length described above, and it is given to the device (possibly to GPU if it is present).
Loss function and optimizer are then defined, Cross Entropy for the loss function and Stochastic Gradient Descent, depending on the learning rate and the momentum chosen during the tuning, for the optimizer.
Since the algorithm provides a checkpoint function, at this point it loads the checkpoint if there is one saved, and if the checkpoint parameter is set to "true". This is made in order to have the possibility to subdivide the phase of training in more steps (in the case in which it takes too much time to make it in an only tranche). Besides the model trained until that point, it loads also until the point to which it had arrived the number of epochs, the value of the loss function and the optimize.
The iteration step is the following, until it reaches the last epoch, and for all the batch in the data-loader:

1. Iterates the data-loader: at iteration i-th extracts the batch and invokes the forward function of the neural network on the data and the length of data (for the batch considered). From a practical point of view, an instance of training is made on a single mark during as much frames as the size of the window.
   Thanks to the collate function described in 5.3.7, the data-loader extract the data, the labels, the number of trials and the length of the data, invoking then the forward function of the Net on the data and the length of data, returning the output of the network ([number of marker $\times$ batch size] probability matrix, see figure 5.13).

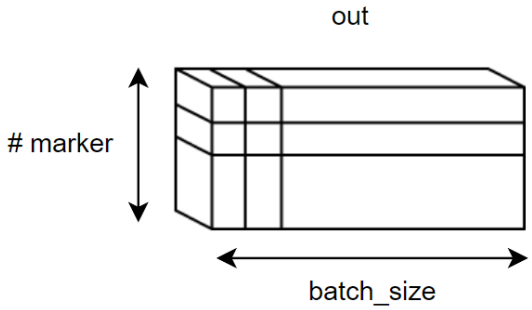


Figure 5.13: Output of the LSTM.

2. Calculate loss through Cross-Entropy (comparing out and labels).
3. Computes the derivative of the loss through back-propagation.
4. Update weights using Stochastic Gradient Descent and the Momentum.
5. Saves the learning state as a checkpoint.

While the iteration is concluded, it returns the trained network.

5.3.10 Input of an unlabeled 3D file

The input of a raw unlabeled 3D file, in .c3d or .tdf format, is performed in a similar way as the one seen for the labeled training files. The main difference is that the tensor is not reordered based on the marker-set, since there are no labels to consider. An interpolation (for gaps < 4 frames) and filtering are then applied to the tensor. If the shape of the markers is smaller than the number of marks reported in the .xml file, horizontal planes (x,y,z coordinates $\times$ n.frame) are generated with nan values. The window of the data is again applied with the same function but on a single tensor.

5.3.11 Prediction on an unlabeled 3D file

The first step in the prediction of an unlabeled 3D file is to initialize the pre-processing function and the data loader. The window of probability showed before, that is the structure of the output of the neural network, is than initialized and filled with values equal to zero.
The algorithm than iterates the data-loader: at iteration i-th extracts the batch and invokes the forward function of the neural network on the batch, applying the forward function.

The prediction phase takes the indices of the maximum values along axis 1 (i.e. takes the number of the marker with maximum probability on each sample of the batch) and returns outputs equal to the Softmax(out), to obtain the relative probabilities of those values. Then inserts outputs into the window of probabilities.

5.3.12 Post-processing

The predict function that uses the neural network is then applied to the tensor and returns a 2D matrix. Each cell of the matrix contains the probability (probWindow(i,j)) for which the j-th windowIdx is labeled with the i-th marker (of the complete marker-set). Each cell of probFrame(m, ms, i1) will contain the probability that the m-th marker is labeled as the ms-th marker at frame i1.

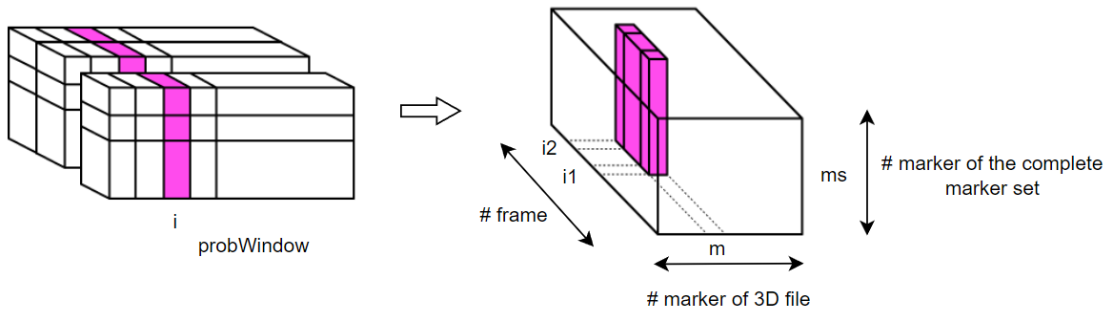


Figure 5.14: Average of Probabilities for Single Marker.

At this point the algorithm looks for frames in which markers appear and disappear. In the 3D tensor, when a marker is occluded it will have (at this marker) the value nan for a few frames, then reappear representing values other than nan. The algorithm takes into consideration all the frames (marker by marker) where there is this switch of values that are placed in a single list and sorted in ascending order.

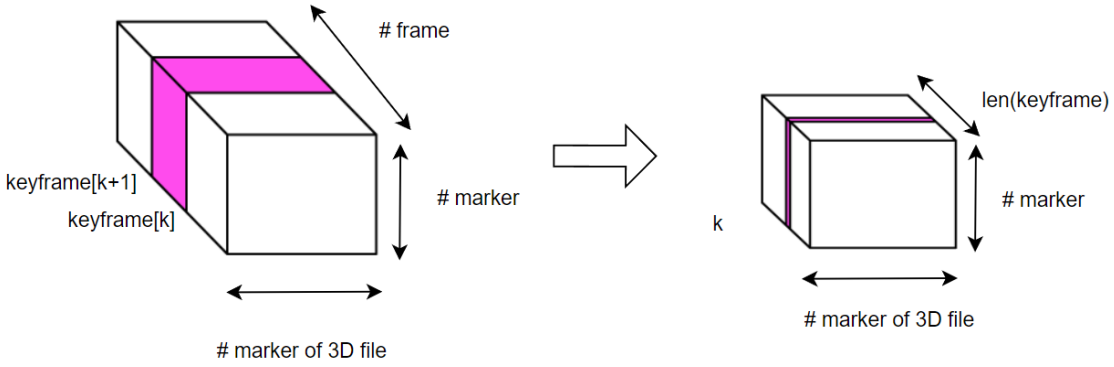


Figure 5.15: Window of Probability of a Single Marker.

For each k-th frame in which at least one marker appears/disappears and its subsequent k+1-th frame in which at least one marker appears/disappears, the mean of the probabilities along the time axis (frame) is calculated.
For example, from a shape tensor (number of markers in analysis $\times$ all markers $\times$ instant $k-th$ $\times$ instant $k+1-th$) The algorithm gets a 2D matrix of shapes (number of markers in analysis, all markers) with the averages of the probabilities. This is done for each pair of consecutive key-frames. At this point all frames have been reduced to a few key-frames using the probability averages of the frames that each key-frame represents. Averaging these values ensures that no markers appear or disappear in these frames.
This happens because from the trajectories whose value is set to nan, the model predicts in a not reliable way the probabilities of the labels on some frames. That is why for these frames the algorithm works with the averages.

For the assignment of the new windows is used the so-called "Hungarian Algorithm" a combinatorial optimization method that solves in polynomial time the assignment problem (already named in section 2.1). Such an algorithm looks for the assignment between the markers of the complete marker-set and the markers of the c3d, in such a way as to maximize the probability. For every key-frame (i.e. representation of more frames) the algorithm looks for an optimal assignment between the markers of the 3d file and those of the complete marker-set. The assignment that maximizes the probability is performed with Hungarian Algorithm.

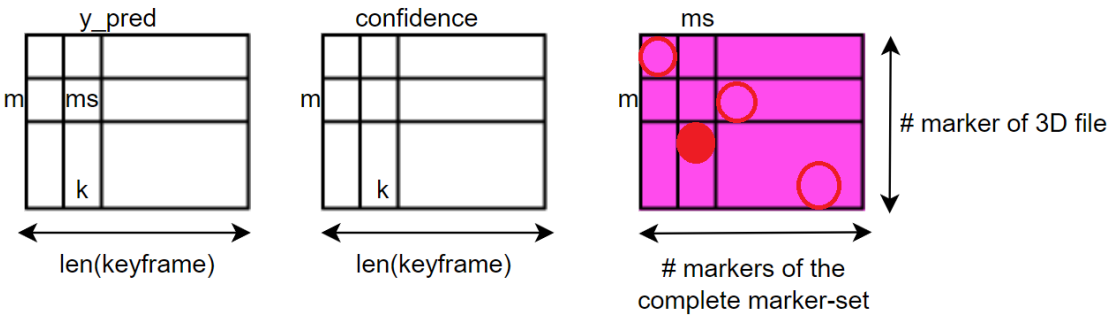


Figure 5.16: Application of the Hungarian Algorithm.

From the assignment it gets:

- **y_pred**: an array where each cell represents the optimal assignment of marker m (marker c from 3D file) to marker ms (from full marker-set) at key-frame k. That is, from a label to each marker to each key-frame.

- **confidence**: in a symmetrical way it also saves the confidence percentage of the label assignment.

The next step is to convert key-frame probabilities to frame-by-frame probabilities.
The labels assigned in y_pred are spread across y_pred_frame until the original number of frames is reached. In this way we get a matrix (y_pred_frame) in which each cell represents the value of the c3d marker assigned to the m-th marker of the complete marker-set at a certain frame.
The confidence_frame matrix is like y_pred_frame matrix, but the cells have, instead of the c3d markers, the probability value of the assignment taken directly from prob_frame.

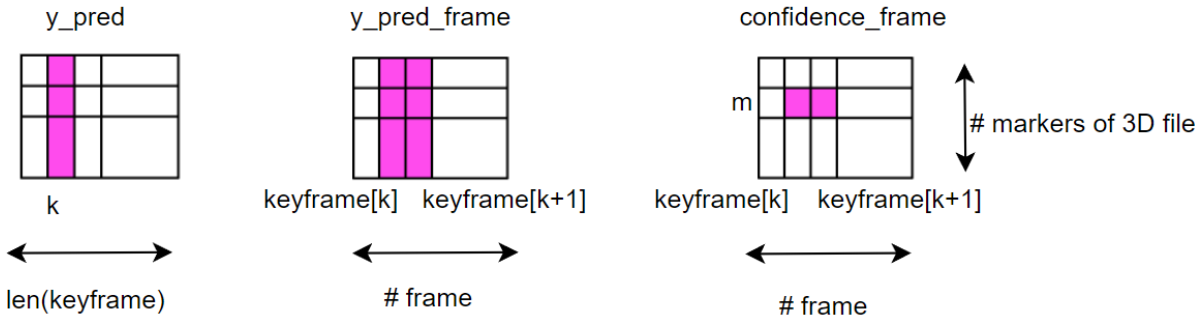


Figure 5.17: Converts key-frame probabilities to frame-by-frame probabilities.

Starting with Y_pred_frame and confidence_frame, a marker from the c3d is uniquely assigned to a marker from the full marker-set by evaluating the assignment frequency for each marker weighted on its probability of being assigned.

At this point the algorithm is close to finishing the procedure of label assignment, but it does a check for possible errors. The algorithm checks if there are duplicate markers. If the trajectories of the two markers are similar then it keeps the duplicate (ghost marker). For each marker in the complete marker-set:

1. If there are duplicate assignments (duplicates): it orders the markers in descending order according to confidence.
2. It checks in pairs (first vs others).
3. If the two trajectories of the markers are very similar (ghost marker situation), then it keeps both markers (they are the same).
4. Otherwise, it sets to -1 in Y_pred those with less confidence.

Finally, at this point, the algorithm returns the labeled 3D file from the raw one given as an input.

6 Results

6.1 Criticality of the Dataset

One of the main criticalities found during the training of the network was related to the nature of the dataset. First of all, a large part of the .tdf files could not be opened in Python. This problem is related to the fact that the BTK (Biomechanical Toolkit) used to open and analyze the files, is not native in Python, and some of its functions are limited. In particular, when the sample frequency of the EMG is not equal to the one of the force platforms, the file simply does not open. The only way to solve this problem is to manually open each file using Smart Tracker and delete the two components, since they are not related to the labeling and therefore are not meaningful in this analysis.

The second problem encountered is that a big part of the acquisitions was not completely labeled, and therefore did not provide necessary information during learning.

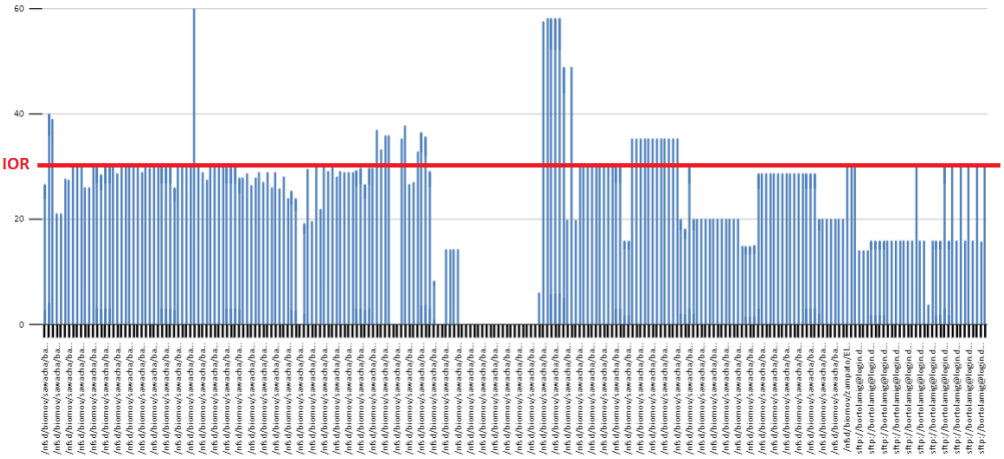


Figure 6.1: Mean number of markers over the Dataset.

As we can see from the Figure 6.1, taken from a pivot table of the .xml file generated during the dataset analysis, only a fraction of files is completely labeled following the IOR marker-set (depicted by a red line), and it is also possible to see some files with more label (more or less the double of the others), that belong to a different marker-set, and so are not usable for the training at all. There is also a

third category of files, the ones with less markers labeled on purpose, which belong to a sub-marker-set taken from the IOR, considering only characteristics from the low body (under the belt).

The third and last problem about the dataset is about those acquisitions that have been started to record too early, or that have been stopped too late, compared to the motor task. In these cases even if the acquisition is labeled correctly, a part of it has to be cut, or the whole file is meaning less for the learning of the network.

Due to these reasons, the available dataset has been greatly reduced from what was originally.

6.2 Hyperparameters Tuning and Training of the Network

Despite the problems encountered, and the limited computational power available, the first training performed on a subset of the dataset has been a fairly success.
This trial has been performed on all the available files related to static movements, the ones in which the subject performs micro-movements because of breathing and balance adjustment.
It's important not to underestimate these files because of the limited movements, and because the analysis is not trivial. This is due to the fact that each subject has different body proportions and different height, the recording has been made in different conditions, and there is always a certain part of noise to take into consideration.
The total number of subjects considered for the first training was 39. From these 39 3D files, with both .c3d and .tdf formats, thanks to the windowing process, the algorithm was able to generate 32.250 different instances, that has been divided in 19350 for the training set and 12900 for the test set. Each one of these instances contained the data related with a single marker during a number of frames as long as the size of the window (in this case 10). This whole process has been described in section 5.3.
The first operation performed on the dataset so created, has been the hyperparameters tuning.
Since the computational power was limited, and to avoid the loss of data, risking an iteration too long, the parameters has been sieved more and more times, trying to retrieve the parameters that offered the best results.
The data were analyzed for a limited number of epochs, obtaining in this way quite high losses. Anyway the purpose was to understand the trend of the accuracy in

order to select the best parameters to insert in the final effective training.
After having found the surroundings of the parameters with greater accuracy, a tuning around each single hyperparameter has been performed, to check how the accuracy and in some case the loss, can change changing the values, even slightly in some cases:

- **Number of LSTM Layers**:
  The number of layers of the LSTM has been screened from 1 to 3, returning a considerable improvement in both accuracy and loss using 3 layers (Fig. 6.2, 6.3).

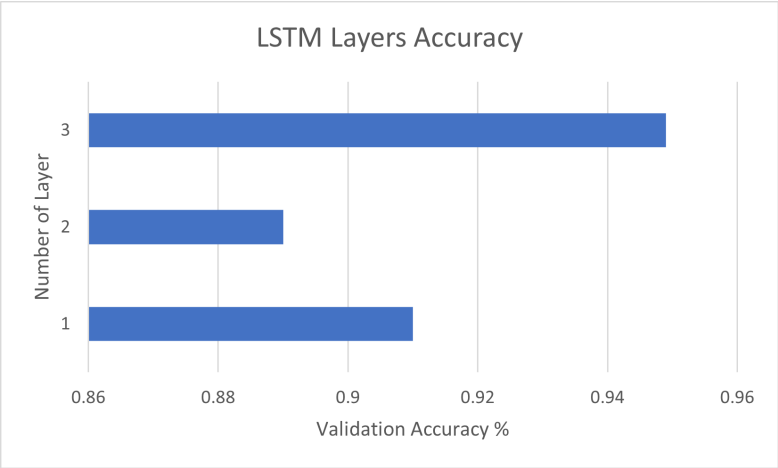


Figure 6.2: Accuracy on LSTM layers tuning.

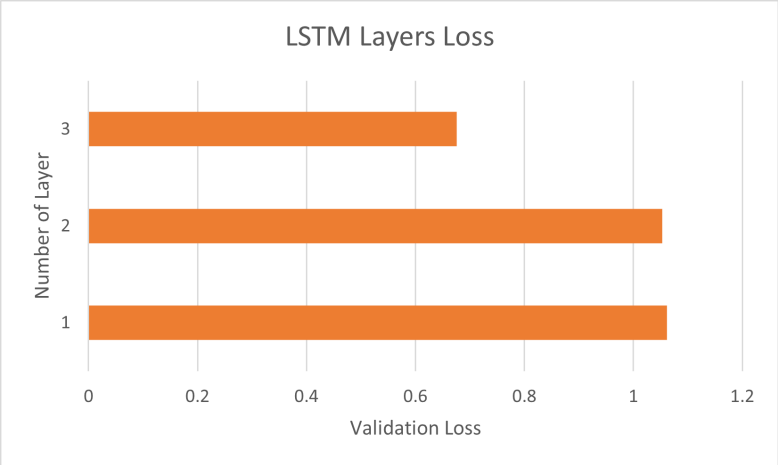


Figure 6.3: Loss on LSTM layers tuning.

- **Number of Cells of Each LSTM Layer**:
  The number of cells of each layer of the LSTM for its architecture, has to be the same. The screening has been performed from 64 to 512 nodes, returning the best performance using 256 nodes (Fig. 6.4). In this case the loss was not so meaningful and therefore is not reported.

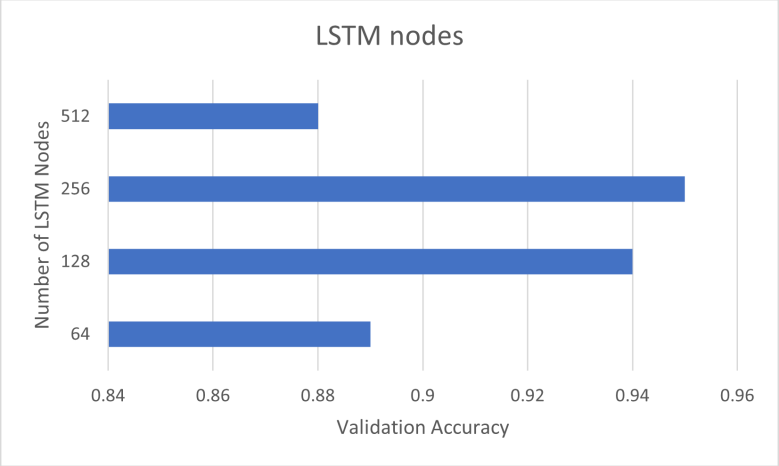


Figure 6.4: Accuracy on LSTM nodes tuning.

- **Number of Nodes of the Fully Connected Layer**:
  The number of nodes of the fully connected layer has been tuned in the same range of the LSTM nodes, from 64 to 256. Even in this case the best result has resulted in the use of 256 nodes (Fig. 6.5). Moreover, the loss was not so meaningful in the choice and therefore is not reported.

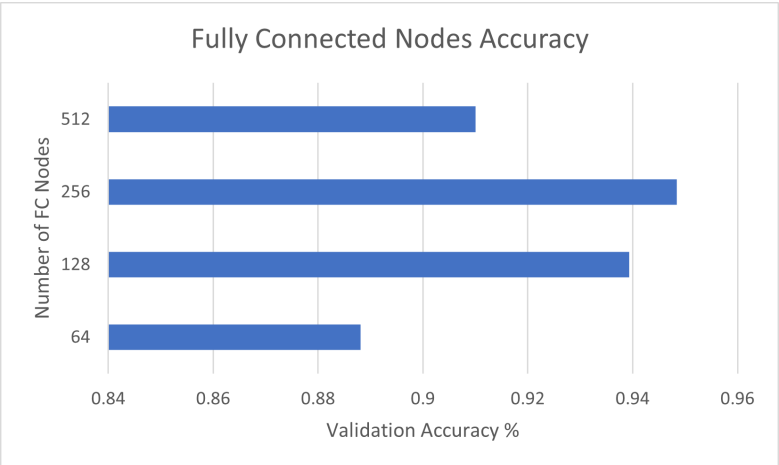


Figure 6.5: Accuracy on Fully Connected nodes tuning.

- **Dropout Percentage**:
  Randomly remove input and hidden units during processing of each pattern, is an efficient way to regularizes each hidden unit to learn not just a good feature, but also a feature that is good in many contexts. Considering that the dropout might interfere with other techniques that also act as regularizers, such the batch normalization used in the network, the value of dropout has been screened in a low range, from 0.15 to 0.19, resulting in a a great result for 0.17 (Fig 6.6, 6.7).

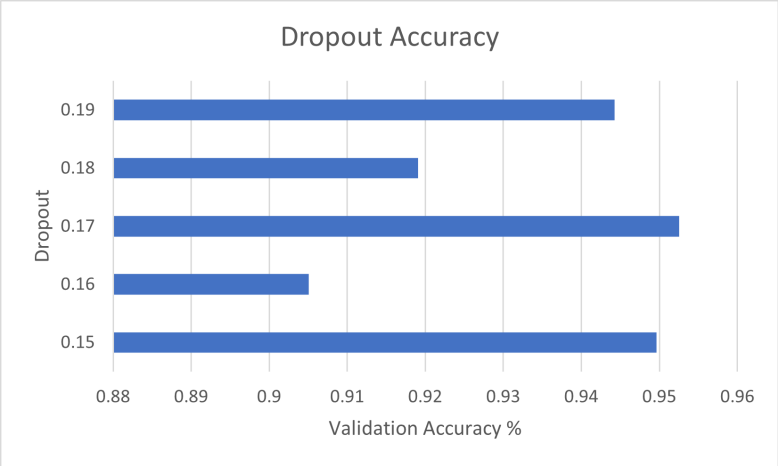


Figure 6.6: Accuracy on the Dropout tuning.

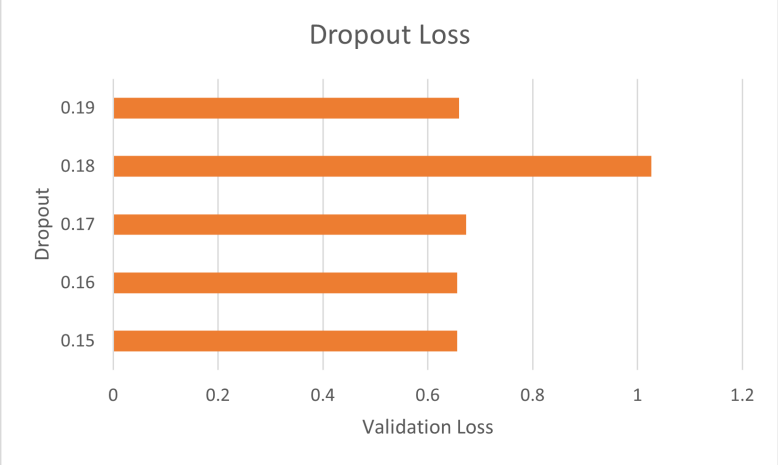


Figure 6.7: Loss on the Dropout tuning.

- **Momentum**:

Adding a Momentum term helps accelerating gradient vectors in the right directions, thus leading to faster convergence of SGD. The Momentum has been screened in a range from 0.5 to 0.9, returning better results for both loss and accuracy using the 0.9 value (Fig. 6.8, 6.9).

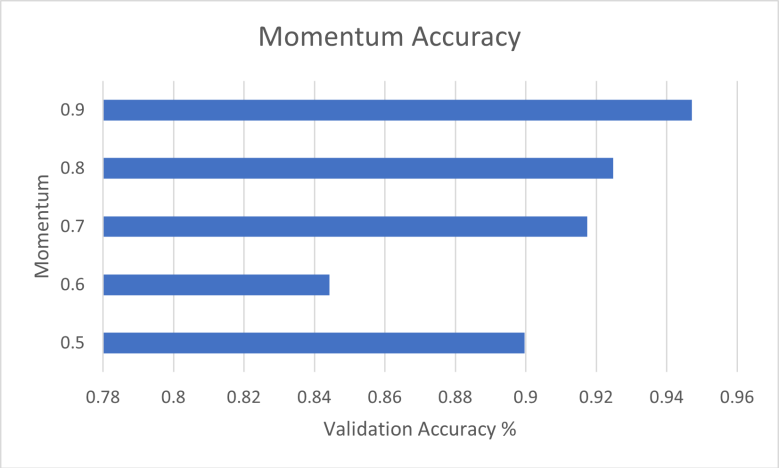


Figure 6.8: Accuracy on the Momentum tuning.

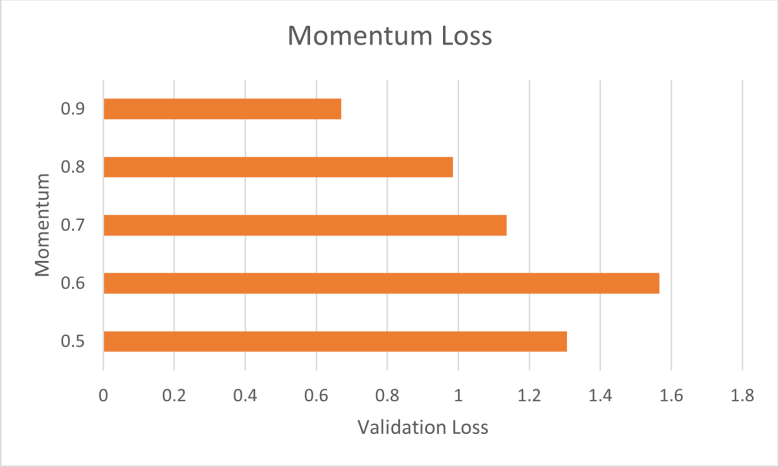


Figure 6.9: Loss on the Momentum tuning.

- **Batch Size**:
  The batch size has been screened in a range from 5 to 50. The results have shown how the performance decreases for values bigger than 30, while the best parameter found for the batch is 5, for both accuracy and loss (Fig. 6.10, 6.11).

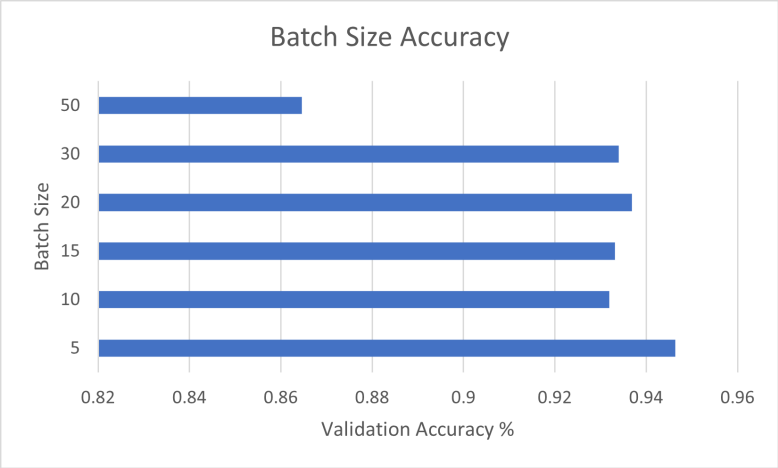


Figure 6.10: Accuracy on the Batch Size tuning.

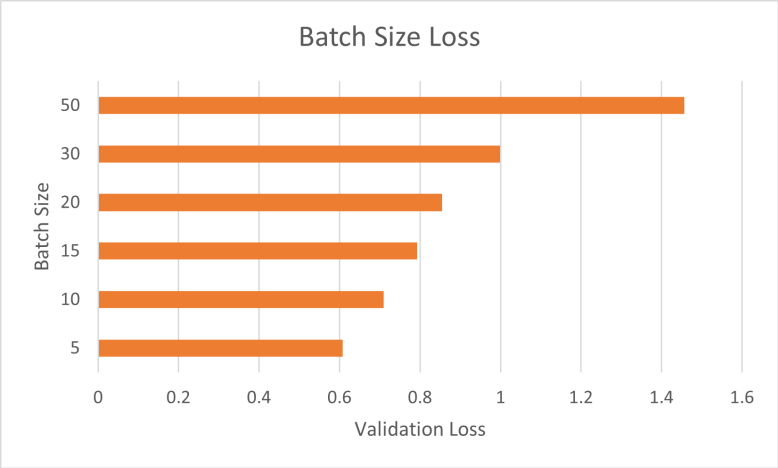


Figure 6.11: Loss on the Batch Size tuning.

- **Learning Rate**:
  The learning rate determines the step size, at each iteration,while moving toward a minimum of the loss function. It can be associated as the speed at which the Neural Network "learns". The screened has been performed in a wide range, more and more times, to get the one that suited more the dataset and the network structure The best value found during the tuning is $1x10^{-4}$, even after having performed an additional tuning in its surroundings values. Both the accuracy and the loss confirm this result (Fig. 6.12, 6.13).

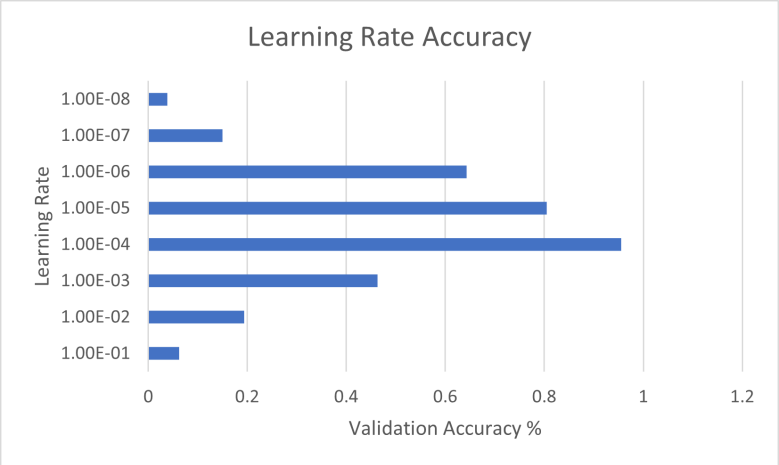


Figure 6.12: Accuracy on the Learning Rate tuning.

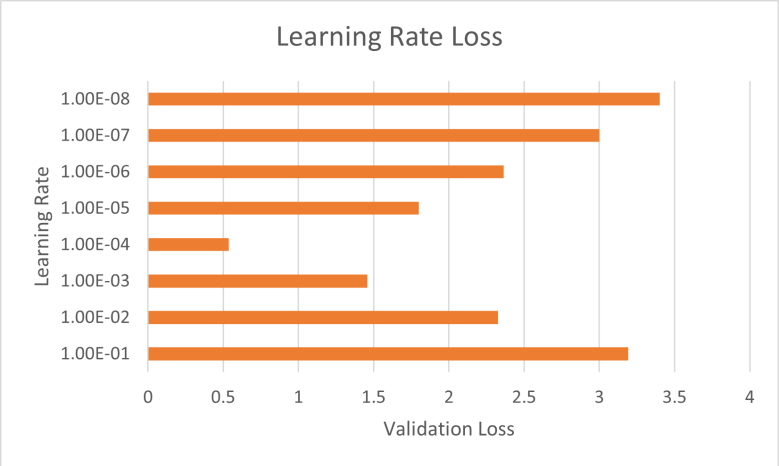


Figure 6.13: Loss on the Learning Rate tuning.

The Training of the Network

As said before, the training has been performed on all the available and suitable files related to static movements. From the 39 3D files, with both .c3d and .tdf formats, thanks to the windowing process, the algorithm was able to generate 32.250 different instances, that has been divided in 19350 for the training set and 12900 for the test set.
The training took several hours and has been performed on a Windows 10 desktop with AMD Ryzen 5 2600X Six-Core Processor 3.60 GHz Cpu, 16 Gb of RAM, and a Radeon RX 5500 XT with 8GB of VRAM GPU.

The training cycle has been performed on 10 epochs, along all the training instances clustered by the batch size selected, and generated during the pre-processing.
The final result has been quite compliant to how expected from the parameters tuning, resulting in a training loss around 0.05, and a test accuracy of approximately 95%.

7 Conclusions and Future Developments

Enlarge and Correct the Dataset

One of the main problems found during this work has been related to the dataset itself, as explained in some of the previous sections.
In the near future, it will be necessary to correct the dataset, removing force platforms and EMG with different sampling frequencies, since the markers are the only parameters in which the algorithm is interested in.
Moreover, it is necessary to resize all those files, whose acquisition has been performed too early or too late respect to the movement analyzed itself, resulting in files that by now are not usable for the training due to the fact they are mostly empty.
Furthermore, the company is acquiring more and more samples in the laboratory or on the sports field, so that the database will enlarge and this will make it possible in the future to have access to a more comprehensive training, possibly with a stronger computational power.

Classification of the Motor Tasks

An interesting feature that would have avoided a time consuming analysis of the dataset, should have been the implementation of a motor task classification, besides the markers one. This classification should be very useful in the future to continue to properly catalog and analyze the dataset available from the company.

Implementation of Attention Mechanism

Even if LSTM neural networks are a great milestone in the field of sequential models, like other models, are also not perfect.
The bigger drawbacks are the long training times, the large memory requirements, the impossibility of parallel training, and others. Because of this, new improved techniques and models were then developed, and one popular approach is the attention mechanism [52].
A future development on this work can be the reconsideration of pre and post processing, with the implementation of an attention mechanism, based on different levels of representation, adding an encoder and a decoder features to the vanilla LSTM used.

Marker-Less Models

Although analysis using motion capture is fairly efficient, the process of marker positioning on the field is time consuming and subject to human error. Furthermore, markers are too easy to occlude during the movement, and in most acquisitions some marker misses for more and more frames.
In the future there will be the needing to implement a marker-less algorithm, which will definitely requires more computation and an AI algorithm more complicated, but at the same time will reduce the overall time consumed, which is the main drawback of the motion capture, in both the application of the markers and in the software for their triangulation, labeling and tracking.

Bibliography

[1] https://geekinsider.com/breif-history-motion-capture/.

[2] https://en.wikipedia.org/wiki/Motion_capture.

[3] https://www.lifescienceindustrynews.com/future-watch/the-role-of-motion-capture-in-biomechanical-science/.

[4] https://motionanalysis.com/blog/infographic-motion-capture-for-biomechanics/.

[5] S. Ghorbani, A. Etemad, and N. F. Troje, "Auto-labelling of markers in optical motion capture by permutation learning," in *Advances in Computer Graphics* (M. Gavrilova, J. Chang, N. M. Thalmann, E. Hitzer, and H. Ishikawa, eds.), (Cham), pp. 167–178, Springer International Publishing, 2019.

[6] D. Holden, "Robust solving of optical motion capture data by denoising," *ACM Trans. Graph.*, vol. 37, jul 2018.

[7] L. Herda, P. V. Fua, R. Plänkers, R. Boulic, and D. Thalmann, "Skeleton-based motion capture for robust reconstruction of human motion," *Proceedings Computer Animation 2000*, pp. 77–83, 2000.

[8] Q. Yu, Q. Li, and Z. Deng, "Online motion capture marker labeling for multiple interacting articulated targets," *Computer Graphics Forum*, vol. 26, 2007.

[9] M. Loper, N. Mahmood, and M. J. Black, "Mosh: motion and shape capture from sparse markers," *ACM Trans. Graph.*, vol. 33, pp. 220:1–220:13, 2014.

[10] S. Holzreiter, "Autolabeling 3d tracks using neural networks," *Clinical biomechanics (Bristol, Avon)*, vol. 20, pp. 1–8, 02 2005.

[11] J. Meyer, M. Kuderer, J. Müller, and W. Burgard, "Online marker labeling for fully automatic skeleton tracking in optical motion capture," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5652–5657, 2014.

[12] T. Schubert, A. Gkogkidis, T. Ball, and W. Burgard, "Automatic initialization for skeleton tracking in optical motion capture," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 734–739, 2015.

[13] S. Han, B. Liu, R. Wang, Y. Ye, C. D. Twigg, and K. Kin, "Online optical marker-based hand tracking with deep labels," *ACM Trans. Graph.*, vol. 37, jul 2018.

[14] J. Maycock, T. Röhlig, M. Schröder, M. Botsch, and H. Ritter, "Fully automatic optical motion tracking using an inverse kinematics approach," 11 2015.

[15] A. L. Clouthier, G. B. Ross, M. P. Mavor, I. Coll, A. Boyle, and R. B. Graham, "Development and validation of a deep learning algorithm and open-source platform for the automatic labelling of motion capture markers," *IEEE Access*, vol. 9, pp. 36444–36454, 2021.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," 1995.

[17] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[18] F. Gers, "Learning to forget: continual prediction with lstm," *IET Conference Proceedings*, pp. 850–855(5), January 1999.

[19] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation*, vol. 12, pp. 2451–2471, 10 2000.

[20] A. Graves, A. rahman Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, 2013.

[21] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.

[22] `https://ai.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html`.

[23] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016.

[24] https://bgr.com/tech/ios-10-siri-third-party-apps/.

[25] https://www.theverge.com/2017/8/4/16093872/facebook-ai-translations-artificial-intelligence.

[26] https://newatlas.com/microsoft-speech-recognition-equals-humans/50999.

[27] A. R. Voelker, I. Kajić, and C. Eliasmith, "Legendre memory units: Continuous-time representation in recurrent neural networks," in *NeurIPS*, 2019.

[28] https://opensim.stanford.edu/.

[29] https://simtk.org/.

[30] https://docs.qualisys.com/getting-started/content/10_how_to_prepare_your_subject/marker_sets.htm.

[31] A. Leardini, Z. Sawacha, G. Paolini, S. Ingrosso, R. de O. Nativo, and M. G. Benedetti, "A new anatomically based protocol for gait analysis in children.," *Gait & posture*, vol. 26 4, pp. 560–71, 2007.

[32] A. Cappozzo, F. Catani, U. D. Croce, and A. Leardini, "Position and orientation in space of bones during movement: anatomical frame definition and determination.," *Clinical biomechanics*, vol. 10 4, pp. 171–178, 1995.

[33] Z. Sawacha, F. Spolaor, G. Guarneri, P. Contessa, E. Carraro, A. Venturin, A. Avogaro, and C. Cobelli, "Abnormal muscle activation during gait in diabetes patients with and without neuropathy.," *Gait & posture*, vol. 35 1, pp. 101–5, 2012.

[34] https://www.c3d.org/.

[35] https://trama.deib.polimi.it/allegati/SmartAnalyzer_handbook.pdf.

[36] http://biomechanical-toolkit.github.io/mokka/index.html.

[37] https://www.bb-sof.com/#about-section.

[38] C. Tomasi and T. Kanade, "Shape and motion from image streams: a factorization method—part 3 detection and tracking of point features technical report cmu-cs-91-132," 11 1999.

[39] https://www.btsbioengineering.com/it/prodotti/smart-dx-2/.

[40] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[41] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[42] J. Balcazar, R. Gavalda, H. Siegelmann, and E. Sontag, "Some structural complexity aspects of neural computation," in *[1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pp. 253–265, 1993.

[43] K. ichi Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural Networks*, vol. 6, no. 6, pp. 801–806, 1993.

[44] A. M. Schäfer and H. G. Zimmermann, "Recurrent neural networks are universal approximators," in *Artificial Neural Networks – ICANN 2006* (S. D. Kollias, A. Stafylopatis, W. Duch, and E. Oja, eds.), (Berlin, Heidelberg), pp. 632–640, Springer Berlin Heidelberg, 2006.

[45] `https://medium.com/analytics-vidhya/long-short-term-memory-networks-23119598b66b`.

[46] `https://www.cuelogic.com/blog/role-of-python-in-artificial-intelligence`.

[47] `https://djangostars.com/blog/why-python-is-good-for-artificial-intelligence-and-machine-learning/`.

[48] B. Michaud and M. Begon, "ezc3d: An easy c3d file i/o cross-platform solution for C++, Python and MATLAB," vol. 6, no. 58, p. 2911.

[49] `http://biomechanical-toolkit.github.io/docs/Wrapping/Matlab/index.html`.

[50] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 448–456, JMLR.org, 2015.

[51] `https://www.ray.io/ray-tune`.

[52] `https://medium.com/analytics-vidhya/long-short-term-memory-networks-23119598b66b`.